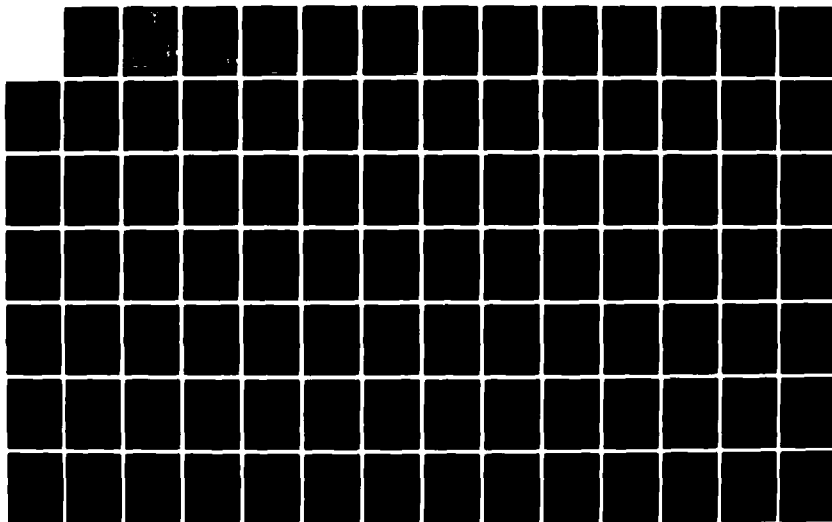
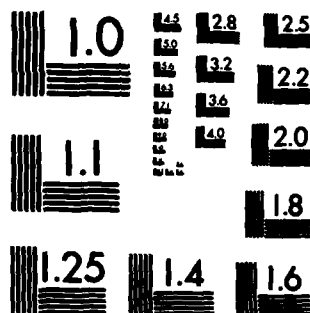


AD-A127 318 DESIGN AND IMPLEMENTATION OF A GENERIC COMPUTER NETWORK 1/3
SIMULATION SYSTEM(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
UNCLASSIFIED S J FOSTER MAR 83 AFIT/GCS/EE/83M-2 F/G 9/2 NL





AD A127318



DESIGN AND IMPLEMENTATION OF A
GENERIC COMPUTER NETWORK SIMULATION
SYSTEM

Thesis

AFIT/GCS/EE/83M-2 Stewart J. Foster
Capt USAF

DTIC FILE COPY

DTIC
ELECTE
APR 28 1983
S
E

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

Information has been approved
for public release and value for
distribution is unlimited.

88 04 28 114

AFIT/GCS/EE/83M-2

DESIGN AND IMPLEMENTATION OF A
GENERIC COMPUTER NETWORK SIMULATION
SYSTEM

Thesis

Stewart J. Foster
AFIT/GCS/EE/83M-2 Capt USAF

DTIC
ELECTE
S APR 28 1983 D
E

Approved for public release; distribution unlimited

AFIT/GCS/EE/83M-2

DESIGN AND IMPLEMENTATION OF A
GENERIC COMPUTER NETWORK SIMULATION SYSTEM

by
Stewart J. Foster

Thesis
Prepared in Partial Fulfillment of the
Requirement for the Degree of
Master of Science

Graduate Computer Systems
March, 1983

School of Engineering
Air Force Institute of Technology
Wright-Patterson Air Force Base,
Ohio



Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

Preface

The purpose of this study was to design and implement a system to simulate computer networks. It is generic in the sense that many different types of computer networks may be modeled with this simulation system. Major categories of networks such as mesh, ring, and bus networks may be simulated by this program.

SLAM, a Simulation Language for Alternative Modeling, was chosen as the implementation language for the program. It offers much power and flexibility to the programmer, especially in the combined Network-Discrete Event orientation that is offered by SLAM.

The objective of producing a program capable of satisfying someone interested in modeling any kind of computer network is very elusive. I have satisfied a portion of that objective.

I would like to express my thanks to three faculty members, Mr. Daniel Reynolds, Dr. Thomas Hartrum, and my advisor, Major Walter Seward, for their advice, encouragement, and patience through an extended thesis effort. Also, the unfailing love, support, (and sometimes distraction), of my wife, Sue, and my children, Tina, Scott, Patty, and Sharon, helped me endure and complete this graduate program.

Contents

	Page
Preface	ii
List of Figures	v
List of Tables	vii
Abstract	viii
I. Introduction	1
Single Computer Facility	2
Centralized Computer Networks	2
Distinguishing Characteristics of Interest in Computer Networks	3
Structures and Techniques Used in Computer Networks	6
Computer Network Design	15
Objective	17
Approach	18
Assumptions/Limitations	22
Sequence of Presentation	23
II. Network Design and Modeling	25
Performance Metrics	26
Design Methodologies	27
General Programming Versus General Simulation Languages	33
SLAM	35
Summary	43
III. Simulation Program Design	45
Structured Analysis and Design Technique	45
Program Design	47
Summary	58
IV. Program Specification	60
User Visibility	61
Network Building	62
Discrete Event File Manipulation	65
Input File Preparation	67
Discussion of Options	69
Summary	86

V.	Slam Network Structures	Page 87
	General	87
	Sequence	87
	Program Connectivity and Slam Network Flow	88
	Diagram Descriptions	91
	Summary	113
	Appendix: Functions of EVENT Subroutines . . .	117
VI.	Implementation	121
	Sequence of Development	121
	Problems	123
	Models Chosen for Demonstration	128
	Summary	129
VII.	Results	130
	Mesh Model	130
	Ring Models	132
	Bus Model	133
	Summary	134
VIII.	Conclusions and Recommendations	135
	Conclusions	135
	Recommendations	136
	Bibliography	140
	Appendix A: Fortran Subroutine Documentation	142
	Appendix B: Quick Reference List of Specifications . .	173
	Appendix C: Demonstrations of Computer Network Models	177
	Appendix D: General Legend of SLAM Variables	198
	Vita	201

List of Figures

<u>Figure</u>		<u>Page</u>
1	Computer Network Communication Subnet	3
2	Performance Measures	25
3	Illustration of EVENT and ENTER Nodes	40
4	SADT Activity Diagram	46
5	Computer Network Simulation	48
6	Simulate Network	49
7	Generate Traffic	51
8	Transfer from Host to Node	52
9	Prepare for Transmission	53
10	Transmit Packet	54
11	Receive Packet	55
12	Transfer from Node to Host	56
13	User Model Building	63
14	Example Network and Input File	68
15	Distributed Network Routing Algorithm Data Structures	74
16	SLAM Network Introduction	89
17	Host Message Generation	92
18	Branching For Creating Next Message	94
19	Processing For Entry to Node	95
20	Branching of Packets Into Node	97
21	Node Acceptance of Packets	99
22	Server Queueing and Starting	100
23	Reentry Into Network For Packet Transmission	101

		<u>Page</u>
24	Transmission Structure	103
25	Entry of Packets into the Retransmission Process	106
26	Hold Queue Selections, Entry, and Time-out	107
27	Message Reassembly Files	108
28	Flow of Completed Messages from Node to Host	110
29	Initiation, Selection, and Cycling of Ring Model Controllers	111
30	Bus Arbitration Controllers(s)	114
31	Statistics Cyclers	115
32	Mesh Model Delay with 0% Error Rate	181
33	Mesh Model Delay with 5% Error Rate	182
34	Mesh Model Throughput	183
35	Delay vs Throughput for Token Ring Model	188
36	Delay vs Throughput for Slotted Ring Model	191
37	Delay vs Throughput for CSMA/CD Bus	195

List of Tables

<u>Table</u>		<u>Page</u>
1	Specifications Based on Topology Choice . . .	72

Abstract

A generic approach was used in modeling and simulating computer networks. The primary type of computer networks of interest in this study are characterized by a communications sub-network of nodes which serve host processors. Local area networks are also considered and may be modeled with this program. All models included packet switching and can be characterized as having distributed, ring or bus topology. The top level of the simulation program design is as general as possible. The lower levels of the design are the building blocks of particular models. The simulation program was implemented with Simulation Language for Alternative Modeling (SLAM). The network and discrete event orientation of SLAM were combined in this simulation system. In general, the SLAM network portion models the computer network components and the Fortran subroutines provides details which define the protocols of the model. Four computer networks were modeled to demonstrate the simulation system. The system is very general. However, many networks may not be modeled precisely enough for formal validation without further development. Further development of simulation systems such as this should be in the discrete event orientation.

DESIGN AND IMPLEMENTATION OF A GENERIC COMPUTER NETWORK SIMULATION SYSTEM

I. Introduction

There is a great diversity in the classifications and definitions of computer networks. Therefore, considerable effort must be made to define the subject of computer networks. One may attempt to build a concept of what a computer network is by adopting the user's view of a computer network. Unfortunately, that view can be distorted by the specific implementations of computing facilities. Also, the structure of computer networks is intentionally hidden from the user. Therefore, a discussion of both computing and communication facilities is necessary to properly define computer networks which consist of both.

This introductory chapter begins with a description of a single computer facility and a centralized computer network. These descriptions lead to an exposition of two characteristics which are considered essential to the definition of a computer network in this paper. Structures and techniques used in computer networks and computer network design are covered in general but in fairly extensive detail. A discussion of the thesis objective, approach, and assumptions completes the chapter.

Single Computer Facility

The view of a single computer facility is a historic concept of what a computer is and how it is used. When the cost of a computer dominated the thinking of system designers, the computer was thought of as the center of the system. All peripherals were connected directly to the central computer for the transfer of data. The peripherals were all in the same general physical area. Thus, no significant communications tasks were required until the first network concepts were introduced in a centralized computer network.

Centralized Computer Networks

Centralized computer networks are characterized by one processor which is at the center of the system as in the single computer facility. The peripherals in this case are remoted by modems and telephone lines of some kind. This represents the first combining of computing and telecommunication resources. The result of this combination is the sharing of the power of a processor among many users. Additionally, the users aren't required to physically go to the central facility for access to this shared processing power. A disadvantage of this level of development is that the processing tasks and communications tasks are centered on the one processor. The development of the computer network concept has grown from this level and now requires additional distinguishing characteristics.

Distinguishing Characteristics of Interest in Computer Networks

The centralized computer network is certainly a network, however, the target of this thesis rests on computer networks that include two additional characteristics. The first characteristic of interest is in networks that have more than one computer that performs the processing tasks. These computers will hereafter be referred to as host computers. This concept of more than one host computer excludes multiprocessor systems. Hosts are independent facilities that are usually separated physically. The variability of that separation can be very large. Additionally, hosts do not share memory as multiprocessors do.

The second characteristic that distinguishes the computer networks of our current interest is the separation of communication tasks from processing tasks. This distinction is a formal separation that is realized by creating a communication subnet that is separate from the hosts which provide the various processing services. A very basic illustration of this concept is shown in Figure 1 which is derived from Davis (6:2).

A primary component of the communication subnet is the network node. The node normally consists of a minicomputer that receives messages from the host computers, stores them, determines the next destination to send them, arbitrates for access to the communications media, transmits them, and checks for errors in all these processes. These processes will be de-

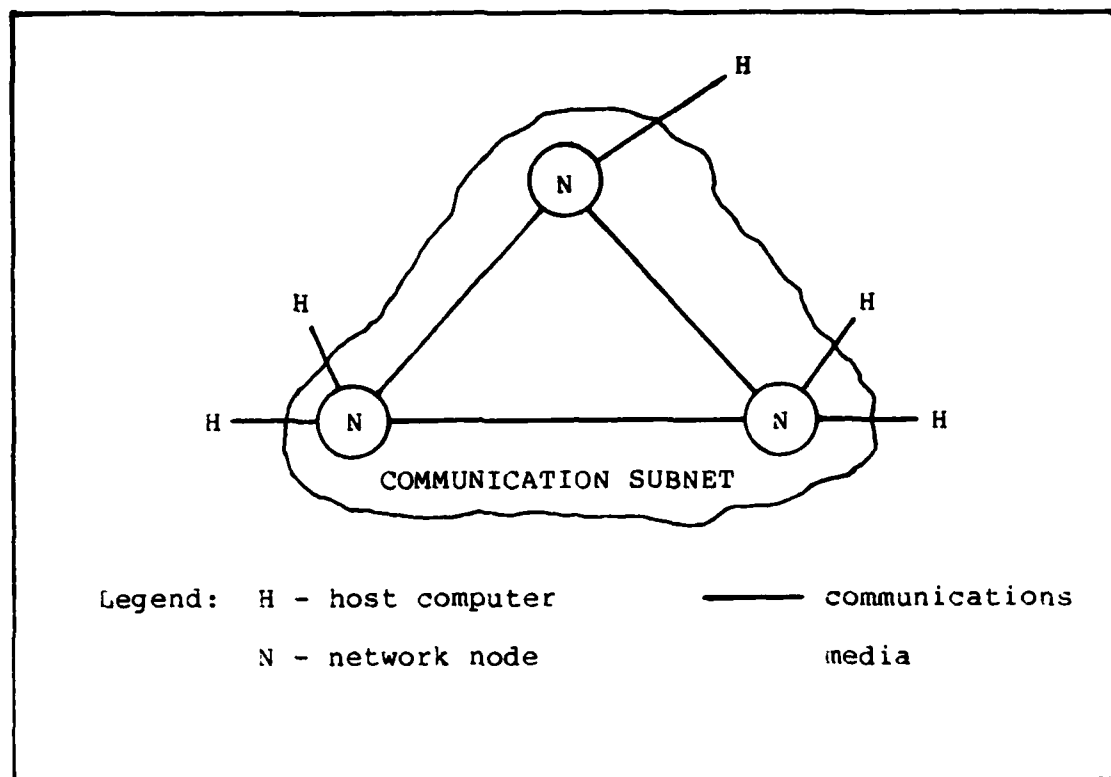


Figure 1 Computer Network Communication Subnet

scribed further in later chapters. However, to complete the illustration in Figure 1, please note that user terminals and other peripherals are normally connected to the hosts. In some cases they can also be connected more directly to the network nodes.

The significance of the separation of computing and communication tasks is best described by noting the functions performed by the hosts and those performed by the subnet. The hosts commonly perform services such as general computing tasks, transaction processing, information storage, and user terminal interfaces. The subnet functions are not as easily

generalized but consist of communication tasks. As a minimum, essential communication tasks performed by the subnet are "overcoming line errors by retransmission or bypassing failed parts of the network by rerouting traffic." (6:4) The specific tasks cannot be clearly enumerated without an exhaustive discussion of protocols and interfaces.

In a specific implementation, the protocols and interfaces fully define the communication subnet and its functions. Although it is not a detailed discussion, please accept the following definitions abbreviated from Davies (6:467-471).

1. Protocol. Strictly defined procedure for interaction across an interface or through a communication facility.

2. Interface. A boundary between two parts of a system across which the interaction is fully defined. The definition might include a type of connector, signal levels, impedances and timing, the allowable sequences of signals or messages and their meanings. By analogy we speak also of a software interface.

Protocols are more simply, a set of conventions between communicating processes on format and content of messages. The software interface indirectly mentioned by Davies can be the vertical interface between two levels of protocols in a protocol hierarchy. The processes mentioned above that communicate through a protocol are related horizontally to each other. Thus, an interface relates that process to a higher or lower protocol.

The networks of interest in this study have the distinguishing characteristics of more than one host and the identification of a clearly defined communication subnet. Elovitz (7:1-4) makes two further definitions of networks that include these distinguishing characteristics. Elovitz illustrates Computer-Communication Networks and Computer Networks by discussing the user's view in each case.

The user view of the Computer-Communication Network is a collection of several computer systems. The user must know what each host system can do and how to invoke its functions using the host-dependent commands. The user must also specify the host in order to establish a connection to that host through the network. Even though the telecommunications tasks are performed by the network, the responsibility for many tasks rests on the user.

Elovitz defines the user view of a Computer Network as one large computing system. He is not required to know various host operating systems. The actual selection of resources and their invocation are transparent to the user. This most refined view of a computer network requires a network operating system. This is realized through the design and implementation of the protocol hierarchy for the network. The result can be the assumption of nearly all responsibilities by the network except those that the user must define for his application.

Structures and Techniques used in Computer Networks

There are many specifications that must be used to de-

scribe a computer network. Some are parametric, and can be stated as parameters such as line speeds and buffer sizes. Others relate to the structure of the network. Questions relating to the structure of the network and the protocols chosen for a network are the questions of interest in this project. Topology, routing procedures, access techniques, and switching methods are some of these questions which will be expanded upon and used to illustrate a few network characteristics known or in-use today. The following discussion is a non-exhaustive treatment. It is meant only to introduce the reader to some well-known structures and techniques.

Topology. Topology refers to the layout of the communication subsystem. It is the pattern of connection of the nodes and links of a network. A simple topology is that of a ring. In a ring, each node has one link in and one link out to form a closed structure. This connection pattern is also known as a loop.

This can be contrasted with a distributed, also known as mesh, topology. A distributed network is characterized by a set of nodes that are connected in such a way that each node may have several paths in and several paths out. This can result in more than one path between any given node pair.

The bus network is certainly different from the other topologies. It is distinguished by a single communication media. Each host computer or device is connected to that single communications media. The media can be a single coaxial cable as in the ETHERNET (6:183), or a radio frequency as in the

ALOHA network (6:159-162). With the bus structure, each device has the capability of direct access to any other device connected to the media.

Each of these topologies has an effect on the other questions yet to be discussed in the design of computer network, one such issue is routing.

Routing. Routing refers to the decision made in a network node regarding the transmission path for an outgoing message. The routing procedures used in a ring network are very basic. Each node can only receive from one node and can transmit only to one node. As a result, messages may travel in one direction only around the ring and there is no decision required for routing.

The bus network is similiar in that no routing decision must be made. There is only one communications media used for transmissions from all nodes in the bus network. As a result, each node has no decision in regard to routing. Regardless of destination, when a device transmits, each possible destination may receive the message. The distributed network presents a definite contrast to the bus and ring networks with respect to routing.

The distributed network may have more than one path out of each node. This implies that a decision must be made and that different paths can be used between node pairs. Since different paths can be used, no control, or poor control, of routing decisions in periods of high traffic can result in an increased potential for a condition known as congestion in some parts of

the network. Congestion may occur in several contexts. To provide more generality, a broad definition of congestion is required. Congestion is "The condition of a communication network beyond the limit of the traffic which it can readily handle, where there is a reduced quality of service and the network must restrict incoming traffic in order to remain effective." (6:461)

Congestion, once it occurs, is primarily controlled by restricting incoming traffic. However, the discussion of routing and congestion that was started above is related to preventing congestion. When there are decisions to be made in routing, an effective scheme can be an effective tool in preventing congestion and retaining efficient service. When possible, this is definitely preferable to restricting incoming traffic and possibly degrading performance.

The routing decision in a decentralized network can use many approaches. Nearly all of these can be encompassed in decentralized or centralized schemes.

In decentralized routing schemes, the network nodes make each routing decision independently. The basis for the decision may be a minimum of static information or a very complex, dynamic information base. The range of complexity these schemes may take is very great.

A simple decentralized routing scheme is a fixed scheme using only the connectivity information of the network. Routing decisions are made based on a minimum-hop algorithm that ignores the status of the network. A slightly more complex and

adaptive scheme may add local status information such as queue lengths to the connectivity information for the decision base. In that case, decisions would be made on the connectivity information as weighted by the time-dependent status of the node.

More sophisticated decentralized routing schemes use information based on conditions external to the node. When decisions are still made locally by each node, a flow of network status information is required. To accomplish this, control messages containing delay estimates (of some kind) must be exchanged between nodes. Nodes store and use this information for a more complex and changing routing decision base. A well-known example of a network using such a routing scheme is the ARPA network (15:48).

A centralized approach can also be used for routing decisions. This approach normally implies that there is a flow of information to one central location. This is a flow of control information that consists of network status in and routing decisions out. The control information overhead traffic in networks using this approach can be considerable (6:104). As a result, variations of this approach seek to limit the quantity of control information and avoid synchronous generation of control information.

Network Access. The discussion of network access that follows refers to the access to the communications media. In many networks, the servers, or transmitters of data, must compete to gain the exclusive access to the communication

media. The process followed in this competition is often called arbitration. The result of arbitration is the selection of a server which can then begin the transmission process.

Ring networks have many variations of access methods, however, two basic methods will be mentioned. One method uses a division of time called slots or frames. Each slot is of constant length and is long enough to transmit one full-sized packet as defined by the network. The node must transmit that which it received during the previous slot unless that node is the origin for a message that just traversed the entire ring. Therefore, the node may access the network with a message generated at its location when no message was received in the previous slot.

Another access scheme normally used only in ring networks relies on a "token" to determine the outcome of the arbitration process. The token is transmitted around the ring in a round-robin fashion. Only the node possessing the token may access the network with message(s) that it has generated. Otherwise, it may only transmit messages that have been admitted to the network in a previous arbitration process.

The network access scheme in a distributed network is often of little consequence. Messages are queued for a particular server and may be transmitted when the server is idle. There is really little or no arbitration as mentioned earlier.

As in the ring network, the bus network has many variations of network access schemes. However, most are based on arbitration processes that use slots or random delays.

In the slotted bus network, all nodes are synchronized to start at the same time, or at times such that messages reach a destination at the same time. A node may begin transmission only at the beginning of a slot. If it receives a message from a host after the slot start time, it waits until the next slot start time to begin transmission. Some variations of this scheme call for the node to wait not only for the next slot, but for a random number of slots.

The bus network may also base the arbitration process on a random delay only. This is described in an overall scheme in which nodes may transmit immediately if the media is idle. However, when the media is active, it must wait for a period of time before trying again. This period of time is established randomly and the scheme for doing so may vary widely.

In bus networks, there is usually a possibility of collisions. Collisions occur when two or more nodes transmit at the same time and have their signals interfere with each other. This may occur even though nodes check the media for activity before transmitting. Propagation delays from one node to another may prevent a node from detecting another's transmission. If a node begins transmission at one location on the communication media, a second node may find the media inactive if it checks during the propagation time between the nodes. The second node will thus transmit only to have its transmission collide with the first node's transmission.

Many of these networks will call for all transmissions to cease immediately when a collision is detected. The recovery procedures are usually based on the same principles used in the arbitration process.

Switching Methods. Switching is broadly defined by the function of, "giving one network subscriber access to all the others." (6:39) There are three switching methods which will be covered next: circuit, message, and packet switching. There is a historical progression through these methods which begins with circuit switching.

1. Circuit switching. This is the method that is used in ordinary telephone systems and is also known as line switching. All connections or switches that are required for communication between two points are established prior to transmission. When the end-to-end physical connections are complete, the transmission may begin. The circuit that has thus been established is held for the duration of the communication process.

This is not very efficient in a computer network where traffic is usually quite "bursty" and of short duration. The time to establish the circuit can contribute significantly to overhead time. Additionally, the remaining components of the network are denied use of all resources used by the circuit for the duration of that circuit. As a result, the concept of message switching was developed.

2. Message switching. Using this method, a message works its way through the network from link to link, queueing at each node in a store-and-forward fashion. Generally, the message

will be stored on a secondary media at each node. Small computers at each node perform the functions necessary to store and process these messages prior to routing them to the next destination (15:2). (In a computer network context, these small computers are the network nodes.) This process continues until the message reaches its destination. There may be significant processing and queueing delays using this switching method. However, the transmission through the network may begin even though all resources that are required are not available at one time as is required in circuit switching.

3. Packet Switching. This is an extension of message switching in that the messages may be split into shorter units called packets. The packets are the unit of information that travel across the network. Packet switching can be the most effective alternative in computer networks. This is because the packets can be interleaved with different data streams according to their channel capacity (6:52).

The bursty and short nature of traffic on computer networks plays an important part in this interleaving process. Network nodes are also known as concentrators because they perform this function routinely. A concentrator accepts traffic from a number of sources and interleaves traffic to the same destination on a common line. Since each source may have only bursts of traffic to that same destination, that traffic can be concentrated on the line to that destination.

Unfortunately, packet switching can increase the complexity of computer networks. One problem created by breaking messages into packets is the reassembly of those packets back into the original message. This leads to yet another problem called reassembly deadlock. Before describing that, consider a very simplified definition of deadlock. In general, deadlock is a condition in which two processes are waiting for resources that are being held by each other. In deadlocked computer networks, packets are holding buffer resources at one node that are needed by packets that are holding buffer resources at another node. If the buffer resources at the second mentioned node are needed by the packets at the first node, then none of the packets can move, deadlock!

Reassembly of messages requires all the packets that are required for that message to be stored at the destination node until the message is complete. If that node has several messages that are partially reassembled, its buffers may become full. If that occurs, none of the packets that are required for message completion can be admitted to the node. Since the partially reassembled messages cannot be moved without those remaining packets, reassembly deadlock occurs.

Computer Network Design

The preceding discussion is a small subset of the structures and techniques used in computer networks. When one considers a more complete set of options available to the designer, an appreciation grows for the problem of computer

network design. There are a great number of variable parameters that combine with the structures and techniques available to make this design problem one of great magnitude. Large networks magnify the problem and challenge the designers modeling capabilities.

The designer needs formal design methodologies to model networks. These are necessary to map the requirements specifications of a network design problem into a completed network design. Metrics must be chosen and some combination of methodologies must be used to produce a final design that is feasible and cost-effective. The next chapter provides a detailed discussion of metrics and methodologies used in computer network design. The methodologies are seldom used by themselves, but in conjunction with the others. One methodology that is found often in the literature is simulation. It can be used in several contexts. In design development, it can be used to validate concepts. It can be used to test and validate designs based on the other methodologies, and it can provide analysis of alternative design proposals.

With such a rich variety of applications, simulation has great potential for contributing to the design process (12:85-100). For that reason, it is the methodology of interest in this project. Moreover, simulation using a program running on a stand-alone computer requires no test-bed hardware or an existing prototype system. A program capable of modeling,

simulating, and analyzing a design is a valuable tool for the designer.

Objective

This thesis project contributes to the solution of the computer network design problem by producing a generic simulation program. This program is a tool that can be integrated with the other design tools and methodologies on many computer network design projects.

The program is to be designed and implemented so that the user will be able to specify the structures and parameters of the network he/she wishes to model. The implementation provides results that can be used to perform trade-off analysis for different selections of parameters and structures in the model. Additionally, the program provides the vehicle itself or the base for the simulation that is required in the design process. The need to create a new simulation program for each project will be eliminated in many cases.

The program can be used in at least three environments. Students can investigate and demonstrate network concepts in an academic environment. Designers can integrate the program with the other design methodologies in design development. Finally, designers can use the program to validate their final design prior to implementation. In each of the environments mentioned, this program can provide useful analysis of specific networks. Trade-off analysis over selected network parameters and protocols is available from the same program.

Approach

To accomplish the objectives of this thesis, the design phase builds upon some previous work. The implementation is in the Simulation Language for Alternative Modeling (SLAM), using a combined Network-Discrete Event orientation. This orientation of SLAM is capable of modeling the design requirements of this thesis and is covered in detail in Chapter II.

Design. A hierarchial approach to the simulation program design was used similiar to that used by Schneider (13). Schneider developed "VANS, A Resource Sharing Computer Network Design Tool" which was written in SIMULA. It modeled an "ARPA-like" network in its first versions and used a strict hierarchial approach to model different networks. A top-level executive was used to initialize and drive the simulation. The second level modules consisted of six classes of processes such as host processes or node processes. This second level specified which of the third level processes were called and the order in which they were called. The second level was designed independently from any specific network model. The third level modules contained the instructions that actually represented the simulation of structures or processing techniques for the particular model under consideration. The user of Schneider's tool was required to select third-level modules from a library or write them to build a simulation model.

This project was developed along the same principles of hierarchy and independence. The hierarchy is represented

somewhat differently due to implementation considerations discussed in the next section. This project also provides a base for modeling as the library in Schneiders work.

The base for modeling that Schneider provided consisted of three levels of modules. The first two levels were quite general in nature and did not form the structure of a model by themselves. The third level modules created the model. The library of Schneiders work was primarily a collection of third level modules that could be called selectively to create the target model. The user was also given the option to write third level modules that reflected modeling specifications not provided by the library. A utility was described that linked the users third level modules instead of the Schneider library modules into the final load module for execution.

The first versions of Schneiders work dealt primarily with mesh networks. Experimentation was done on several questions that were primarily relevant to mesh networks (13:244).

As a base for modeling, this project provides structures for three major topology types of networks. Structures are provided to allow the user to model mesh, ring, or bus networks. For each of these networks, several choices of easily recognizable or well-known structures are provided. For example, token and slotted controllers for ring networks, and ALOHA and Slotted ALOHA bus networks are available. The user can experiment with these networks for simulation system familiarization prior to developing additional modeling structures.

The simulation system provided by this project is packaged quite differently from the Schneider work. There are SLAM network structures and Fortran subroutines. The functions and hierarchial relationships of the simulation package were influenced significantly by the implementation. The next major section, implementation, covers the system structure in detail.

The design will also identify and isolate system processes as described by Fortier (8). Fortier developed "A General Simulation Model for the Evaluation of Distributed Processing Systems" which emphasized bus networks used for real-time control applications. He identified the message producer, arbitrator, and the user as the main components of his simulation system structure. The message producer simulated the generation of messages and their queueing in output queues. The arbitrator simulated mechanisms for allocating and controlling access to the communication channels. The user simulated transmission, reception, and delays involved for each message.

This project uses similar structures to simulate system processes. A process for routing in distributed networks is one that is required although Fortier didn't use one because he emphasized bus networks. Additionally, the processes used by Fortier are broken down to allow more generality. The details of that division in this project are, once again, affected by implementation considerations.

Implementation. The SLAM implementation of this project combines the network and discrete event orientations. A detailed description of the orientations, or views, available

in SLAM are given in the next chapter. However, the network portions of this project use entities, attributes, queues and other structures as found in GPSS and Q-GERT (11:434-447). The discrete event portions of the program are coded as Fortran subroutines and functions.

The hierarchy of program modules referenced earlier is affected by this division of modeling into SLAM network components and Fortran coded discrete events. In general, computer network processes that are independent of specific network models are represented in the SLAM network components of the program. The structures of the network model that are to be variable or analyzed are represented in the Fortran discrete events.

The SLAM network is broken into generalized structures that can be used to build the overall model of hosts, nodes, and servers. From this network, the Fortran subroutines are called. The SLAM network may directly call events and user functions.

The events are high level modules which may call into several levels of other subroutines and functions. The events are in some cases a single subroutine representing a protocol option. In other cases, the protocol options are actually present in the lower level subroutines which are selectively called from the events.

User functions are used for determining activity times, making miscellaneous assignments, and evaluating logical conditions that the SLAM network cannot. Generally, they provide

simple functions that support the SLAM network and some of the events.

The user may modify or replace discrete event modules to build a target model. However, many choices are implemented with global switches. Thus, the options described further in Chapter IV are available in the base program without substitution of user-coded discrete events.

Parametric specifications are inserted into the model primarily in two ways, replicating network components and inserting specifications in the network statements. For example, the number of hosts at any node can be increased by repeating the network statements that represent a host but with different indexes. Specifications of network node buffer capacity are made via the SLAM INTLC statements.

Built-in Analysis. SLAM data collection and presentation routines are used to automatically provide statistical data on the performance of the network modeled. Data normally presented are message delay, packet delay, network node queue lengths, buffer resource utilization, and server utilization. Time independent or persistent statistics are used where appropriate.

Assumptions/Limitations

1. The target models desired by the users must consist of a computer network as defined earlier. Specifically, the model will be based on more than one host computer, and a

communication subnetwork independent of the hosts must be included in the model.

2. Packet or message switching must also be a part of the users target model.

3. The user must accept responsibility for the following:

a. Specification of parameters

b. Implementation of low-level modules or discrete events that are not provided in the base program. For example, the controlling mechanisms for the token ring and slotted ring are concentrated into one subroutine that is called by a SLAM network cyler. A user may replace that routine to change the overall control of the network. Also, a low-level function is the decision mechanism for packet entry to a node. Replacing that function would be a means of modeling other entry criteria.

c. Generation of functions to dynamically change any specification or communication link during the simulation.

4. User trade-off analysis requirements can be satisfied with automatically provided data.

Sequence of Presentation

The next chapter discussses modeling techniques in a computer network context. It discusses modeling in general and leads into modeling by simulation. Modeling by computer simulation is emphasized to orient the reader to the topic of this thesis. Chapter III provides a high-level view of the logical design of the simulation program. It provides a context

for the discussion of the program specifications provided for the user in Chapter IV. Chapter IV is a detailed presentation of the specific modeling options available in the simulation program. It gives the reader a detailed concept of what the program does. Chapter V is an introduction to the SLAM network structures that were developed in the course of this thesis. The flow through each structure is presented to aid in the conceptualization of the model. Chapter VI describes the implementation phase of the simulation program. It describes problems encountered and the models that were used to demonstrate the program. Chapter VII evaluates the results of the demonstrations. Chapter VIII provides conclusions and recommendations regarding the application potential of this simulation system.

For users interested in further documentation not included in this document, a program and documentation package for this thesis is available from Major Walter Seward, AFIT/EE Faculty. It consists of all Fortran subroutines with a legend of attributes and variables, a SLAM network, description of output, and sample output.

II. Network Design and Modeling

There are several methodologies used in the design of computer networks. Each can be viewed as a tool or technique used by the designer to produce and evaluate a computer network. The effectiveness of the design process depends upon the tools used by the designer and how well they perform. The performance of a design tool in this context can be evaluated by its ability to model a network, and its ability to give some measure of performance.

This chapter describes metrics used to evaluate computer networks and the design methodologies that provide such analysis. This discussion first gives the reader a look at the full inventory of tools used by the designer. This view is given prior to focusing on the design methodology that is central to this thesis, simulation.

Simulation is just one of the methodologies used by the designer. As all the methodologies, it should be integrated with the others in a design effort. The presentation of this chapter begins with the metrics and methodologies used in computer network design. Simulation is the last methodology covered. It is immediately expanded and emphasized by a general discussion of the languages that can be used in simulation. Finally, the specific language chosen for the implementation of this thesis project is described in detail.

Performance Metrics

A significant amount of effort is required in the design of computer networks. Through the use of design tools, a network must be proven feasible with respect to several performance measures. Computer networks are most typically evaluated with respect to throughput, message delay, cost, and security (1:6-1 - 6-4,9-1 - 9-2). Security can be treated at several levels of abstraction but will not be considered directly in this thesis. Therefore, this discussion will be limited to throughput, message delay, and cost.

Throughput is the rate at which finished units of information are produced. Message delay is the average time for a single message to be sent from source to destination. Cost is calculated from the resources used in the network. Stewart (17) defines network response to be the reciprocal of message delay and presents the relationship between the three metrics as shown in Figure 2.

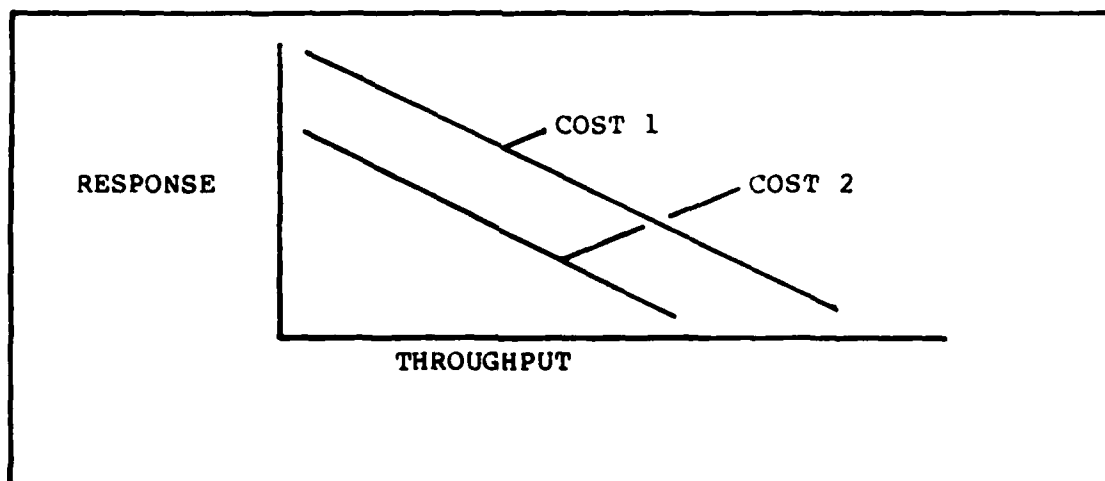


Figure 2. Performance Measures

Stewart illustrates Figure 2 with the following remarks. "At a given cost level, additional throughput can be obtained at the expense of network response and vice versa. Maximum throughput is obtained by ensuring that there will always be a packet waiting in the queue when a circuit is ready. Conversely, maximum response is obtained by ensuring that a circuit is idle when the packet arrives. Both throughput and response can be improved by adding resources, but this increases the total cost of the system. Network models are used to study the relationship between cost, response and throughput for a specific network configuration."

This project will address design efforts that are driven by throughput and message delay constraints. That is, cost levels will be assumed fixed and defined by the structure modeled by the network. The objective of the user is to optimize or balance the other metrics as required by the problem statement.

Design Methodologies

Allen (2) identified five methods for modeling computer networks: rule of thumb, linear projection, analytical modeling, simulation, and benchmarking. Stewart (17) provides a sound discussion of each method. His comments will be used liberally in presenting an alternate and somewhat parallel enumeration of methodologies: intuition, experience, analytical modeling, and simulation.

Intuition. Intuition is usually manifested in rules of thumb. The rules of thumb are usually relatively simple generalizations which may or may not be useful depending on the environment that they are used in. The environment may not respond as expected. For example, network loading may not be allowed to exceed 10% on one network due to degradation of service from competition for resources. However, that loading may severely underutilize a different network with more efficient protocols.

Experience. Experience is also an element that is required in rules of thumb, but is also the basis for linear projection. Using linear projection, a common assumption used is that performance characteristics are directly proportional to physical changes and that changes in one lead to changes in the other. For example, with a circuit of known capacity one may double the capacity of a circuit in an attempt to double its throughput. That will be a correct action if these parameters are independent of the rest of the network. Independence is rare in a case like this because the receiving computer may become overloaded. That, or other factors, could interfere to actually reduce throughput rather than increase it.

Experience when used as a design tool is limited because it requires parameters to be linear and independent. Additionally, it requires a known point of reference in a known environment. As design aids, intuition and experience are similarly limited because they both depend on previous results

and those results are dependent on the particular network being studied.

Analytical Modeling (17). Analytical modeling, which includes both deterministic and probabilistic models, is a powerful and relatively inexpensive method for studying a network. These models use complex mathematical representations of the network to predict message delay times and queue lengths. When both the message interarrival and transmission times are constant, a purely deterministic network model can be constructed. When the interarrival or service times are not constant, it may be possible to construct a queueing model of the network.

The specification of a network queueing model requires detailed knowledge of the network's message population (source of the messages), arrival distribution, queue discipline, queue configuration, service discipline and service facility. Queue discipline refers to the way messages enter or reject a queue and their degree of patience once in the queue. For example, do packets enter the rear of the queue? The middle? Can they refuse to enter a queue if it looks crowded? Or, once in the queue, can packets leave before being served? Queue configuration refers to the physical characteristics of the queues. Are they finite? Infinite? Is there a single queue or many queues for each server? The service discipline refers to the order in which messages are transmitted. Examples are First-In/First-Out (FIFO), Last-In/First-Out (LIFO), Shortest-Message-First, etc.

Service facility refers to the physical characteristics of the transmission equipment, baud rate, number of channels, etc.

The use of analytical modeling is limited because queueing models provide only a generalized view of the network and may be difficult to use in large complex systems. Complexity grows exponentially with the number of network components. Additionally, several layers of protocols can contribute greatly to modeling complexity. As a result, this methodology is usually used in conjunction with other methodologies such as simulation.

Simulation. Three main approaches can be identified when simulation is used in network design. The first is simulation of a new design using an existing network as the simulation vehicle. This has been done extensively on many networks by trying design changes with a representative workload during periods when the network is lightly used or not used at all. This approach may be conceived as an evolutionary process or as a formal extension of trial and error. The continuing literature found on the ARPA network is an example of this method.

A second approach uses a prototype or a special test-bed arrangement of several small computers. These computers are carefully interfaced to simulate the components of a network. CHIMPNET: A Networking Testbed (10) is an example of such a system. It is a relatively inexpensive interconnection of microprocessors for experiments with distributed systems. CHIMPNET was designed using as inexpensive hardware as pos-

sible, simple software, and simple instrumentation interfaces. However, no reports of use were noted yet.

Flexibility is often a limitation of simulation using test beds. Bennett (3) describes a distributed system of cooperating micro computers. A small prototype was considered but not used because it "would not result in a flexible enough test-bed system." Instead, a software emulation was used. In another study, Christopher (5) presents a technique for coordinating the components of a queueing system simulated over a network of microcomputers. A major problem was noted in keeping the microcomputers busy in this simulation method.

These first two approaches to simulation are also known as variations of benchmarking. Limitations are the availability of a network or the hardware to simulate the network, and flexibility. These approaches can be very expensive and may not be considered until the basic concepts of the design that is proposed have been proven by another design methodology.

The final approach to simulation of computer networks in this discussion is that of a simulation program running on a stand-alone computer. The simulation program models the functional characteristics of a network. The program is run by sampling a representative workload or by specifying an approximation of the workloads to produce results that can be analyzed. The design and implementation of the simulation program can be expensive in terms of program development. However, it is often the most practical means of studying complex functional relationships in a large system.

Simulation programs are often written so that different parameters such as number of nodes, number of hosts, number of links, and data rates can be changed and analyzed. In addition to parameters that are changeable, it is also desirable to be able to change and study functional characteristics of the network such as routing algorithms, link protocols, flow control policies, or message protocols. Considerable design efforts are required for simulation programs with capabilities to change such structures. That is also the reason why many simulation programs are not very general and why the more general programs can be expensive to develop.

There are several problem areas noted by Schoemaker (14:71) that the modeler must guard against when using simulation in general. The first is a wrong design. This is a mis-translation of the targeted system to be simulated. The design must map the target system directly into the model developed for the simulation. Second, Schoemaker mentioned errors in coding. This is really a self-evident possibility and is certainly involved in simulations. Finally, the selection of the wrong tools can be a problem in simulation. This is really a discussion of choice of language when using a simulation program. The next major section is a more detailed discussion of this choice.

Summary of Design Methodologies

Each one of the methodologies has inherent limitations when viewed independently. In practice, they are often integrated with each other in design projects. The strengths of

each can be made to complement the others in order to produce an efficient and practical design. Of all the methodologies, simulation on a stand-alone computer is usually cited most often as being one of the tools used.

General Programming versus General Simulation Languages

The choice of languages is an important decision for the implementation of a simulation. There are many advantages and disadvantages to each of the two global choices in this section. The following is a discussion of a subset of those considerations which are outlined by Remes (12).

General Simulation Languages. These languages are program packages specially intended for simulation. They may be based on a general programming language such as FORTRAN or written entirely in an assembly language. They can be used for many kinds of simulation problems and have extra features needed in nearly all simulations. Features such as queue handling, timing functions, priority functions, event processing, and automatic data collection are commonly provided. These features can relieve the programmer from the sometimes overwhelming overhead of tasks involving many data structures, pointers, and file handling.

The simulation language may allow the user to build his model easily, but often not to the level of detail desired. Graphical illustrations that map into specialized constructs of the language often greatly facilitate the model building process. However, the constructs of the language can restrict the modeling options of the user, especially at the lower

levels of detail. A queue may be specified very simply, but distinctive or unusual queueing or service disciplines may not be modeled by the queue structure provided.

Depending on the language, many structures may be hidden from the user. Automatic data collection for instance, can be very helpful. However, if the output mechanisms are not acceptable for the user and not visible either, the user may not be able to generate the analysis products that are needed.

General Programming Languages. In this context, general programming languages are those which can be executed on almost any computer. Using these languages, such as FORTRAN, PASCAL, or ALGOL, the user can do the same as with a simulation language. That is naturally so, because many simulation languages are simply based on the general programming language. What the simulation language could do for the user, however, must all be accomplished by the programmer in this case. The construction of queueing and timing functions may be the most laborious part of the design and implementation of a simulation in a general programming language.

The user is freed from the specialized constructions of the simulation language when using a general programming language. The user may now use the entire set of instructions of the language to construct the model to as low a level of detail as desired.

The collection of statistics may also be done freely and without artificial restrictions. Of course, the user must create his own analysis packages or adapt other packages.

Perhaps the greatest advantage of using a general programming language for simulation is that it is possible to make more efficient programs. The use of computer resources can often be controlled more effectively by the user choosing more efficient data structures and fine-tuning the timing functions to run more efficiently for a particular application. A final fall-out realizable when the use of computer resources is minimized, is the construction of an interactive simulation model.

SLAM

Simulation Language for Alternative Modeling (SLAM) (11) is an advanced FORTRAN-based language. It allows simulation models to be built based on three different world views. These views, or orientations, are network, discrete event, and continuous modeling.

These orientations can be used in combination with each other also. This project uses the combined network-discrete event view. Since continuous modeling is not used, the following sequence of presentation will be the network, discrete event, and combined network-discrete event orientations.

World Views or Orientations.

1. Network. SLAM provides network symbols for building graphical models that are easily translated into computer programs. These are used to construct queueing systems which model a target network. These networks consist of structures to create, queue, and process entities that flow through the system. Entities are the basic unit that represent anything the

modeler desires to simulate by assigning attributes to the entity. The attributes define the entity and can be used to determine the branching that the entity takes through the network. As entities flow through the network, they are constrained by the number of activities (number of servers) and by resources. The number of servers is usually specified in the network. However, the resource capacity is usually declared initially with utilization varying as entities seize and free resources.

At a high level of abstraction, SLAM network structures can be used to define networks that model computer networks. Prior to illustrating the SLAM network structures more specifically, note a simplified development of a portion of a computer network.

Message generation to packet transmission in a computer network can be viewed in this simplified sequence: Messages are generated by host computers or their peripherals. The messages are queued waiting for any required host processing to prepare packets for admission to a network node. After processing, the packets may have to wait in the host until sufficient network node buffer resources are available. When admission to the node is granted, the packets are transferred to the node where they are processed and then queued for a circuit.

Several SLAM node types, activities, and resources can be the primary SLAM network structures for modeling computer networks. The following illustration parallels the generalized sequence of the preceeding paragraph. CREATE nodes create

entities to model the host message generation process. QUEUE nodes file entities as they wait for service activities just as hosts queue messages that require processing into packets. The service activities following the QUEUE nodes require the elapse of time which models the processing time in the host computer. AWAIT nodes then collect and file entities that require a resource just as packets must be held in the host until the computer network node has sufficient buffer resources to accept them. The SLAM network resource controls the release of entities from the AWAIT node to model the control of buffer resource over packets in the computer network. Once entities are released from the AWAIT node, they are committed to another activity which models the transfer between the host and computer network node. Following that activity, entities enter an AWAIT node to model the queue in a computer network node for a server or circuit. A SLAM resource controls the release of entities from the AWAIT node to model the circuit availability in the computer network.

Although just a few SLAM network structures have been illustrated, they are sufficient to model many of the network structures used in this project. The discrete event view is combined with the network view for this project to complete the modeling requirements. Therefore, the discrete event view is presented next.

2. Discrete Event. The discrete event orientation consists of modeling a system by describing the changes that occur in the system at discrete points in time. In this orien-

tation, time does not advance within an event. Changes to the state of the system occur only at event times. Therefore, the behavior of the system is simulated by state changes that occur as time is advanced from event to event.

SLAM allows the state of the system to be changed in four respects. Values of variables may be changed, the number of entities in the system may change, the values of attributes assigned to entities may be changed, and relationships between entities can be changed by manipulating the files. Files are the structure in which entities reside.

The traditional discrete event driven type of simulation is based on a simulation clock. The clock is advanced in time until the next event is scheduled to occur. At that time, the appropriate subroutine(s) are executed to change the system state and schedule future events. The clock is then advanced until another event is scheduled or until a stopping time has occurred.

A simulation system can be built entirely within this orientation. A main "driver" program handles the simulation clock, calls subroutines that represent events, and schedules events. In SLAM, this orientation can be combined with the network orientation by adding two key interfacing structures presented in the next section.

3. Combined Network-Discrete Event Modeling with SLAM

This combined orientation is implemented with a SLAM Network and a collection of Fortran-written subroutines. This project is implemented with two main files. One contains the

SLAM network statements, and the other contains the Fortran subroutines and user functions. These files, each representing one of the orientations, are used together in an overall structure shown in Figure 3a. The main program calls subroutine SLAM which drives the simulation through the SLAM network. The SLAM network, in turn, invokes the discrete events.

Figure 3b illustrates the two key structures used to transition between the two orientations. The transition from the network to discrete event is made with the EVENT node. The transition from discrete event to network is made by entering an entity into an ENTER node.

The network provides the framework for the system state and timing functions. However, when the model demands complex system state changes or changes to an entity that are not implementable in network structures, an EVENT node is inserted into the network. An entity arrival at the EVENT node causes a transfer of control to the corresponding subroutine in the collection of Fortran-written subroutines that represent the events.

The event subroutine may change the system state and may cause additional future events by reentering an entity into the network via the ENTER node. The SLAM enter subroutine is called by an event subroutine to cause an entity to be re-entered in the network at an ENTER node. That entity then flows through the network and causes future system state changes or events through additional EVENT nodes. The modeler is free to insert EVENT and ENTER nodes anywhere in the network. As a result,

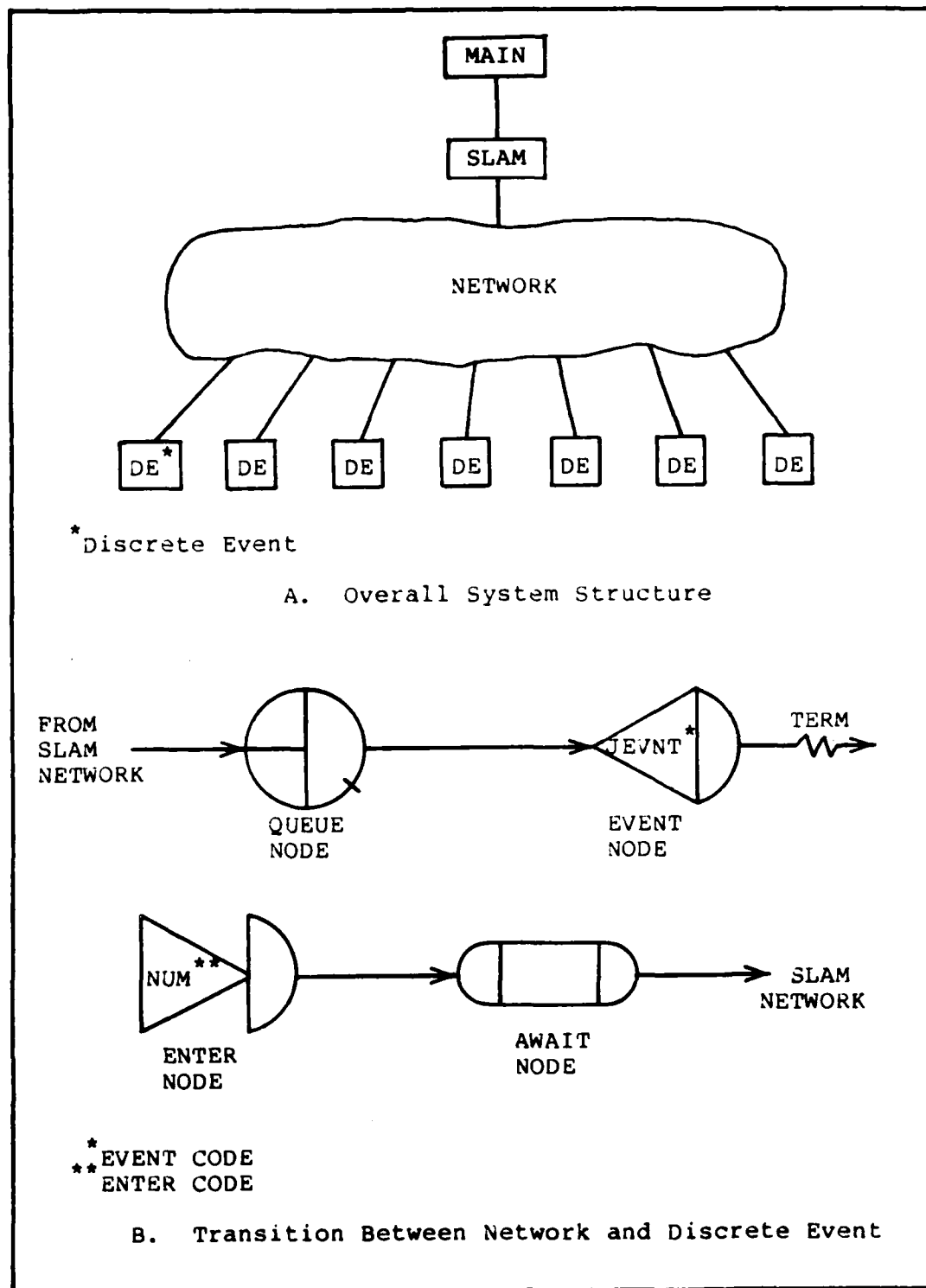


Figure 3. The Combined Network-Discrete Event Orientation

there is a great deal of flexibility offered by the combined network-discrete event orientation of SLAM.

Considerations for Language Selection.

1. General versus Simulation Language. In the development of the proposal for this thesis, two criteria were considered in relation to the choice between general purpose and simulation languages. The first was the accuracy of modeling, and the second was ease of use.

In most cases, a general purpose language supports precise modeling better than a simulation language. However, the SLAM network view, when combined with the discrete event view, can remove the constraints of the special simulation language structures. The portions of the network requiring modeling that is not available in the SLAM structures can be coded directly in Fortran. This is done by including event nodes in the SLAM network to cause the user-written Fortran events to be executed. SLAM is capable of modeling the design required for this project by using the combined network and discrete event views.

The ease of use can be considered at two levels, development of the program and running the program.

All computer network simulation programs must include timing functions, queueing processes, filing, and analysis. The overhead of developing these facilities is avoided by using SLAM. SLAM can handle these functions automatically and with little development effort.

The ease of running a SLAM program is undoubtedly a disadvantage. Since there are many SLAM dimensioned variables

that consume large amounts of memory, the SLAM simulation package is too large to run interactively. As a result, a batch environment must be used.

This batch environment can be improved with two facilities. One that has not been developed, is a program development program written in a general purpose language. This program would be an interactive tool that would accept user parameters and modeling choices to produce the SLAM network statements. The discrete events are written in another file and would not be involved in this process.

The second facility exists in the SLAM system. Multiple runs can be made with SLAM input statements executed between runs. The input statements can be used to clear the system or make variable or state changes. In this way, the system can be reinitialized to different states for each run.

2. Desired Simulation Language Features. The following features are enumerated by Pritsker and will be commented on briefly with respect to SLAM.

a. Training Required. SLAM is relatively easy to learn because of its graphical definition of network structures.

b. Coding Consideration. The translation of the SLAM graphical presentation to code was designed to be simple and straight forward. However the resulting code is not very self-documenting. The discrete event portions of the simulation program are coded in Fortran 77.

c. Portability. SLAM is not that portable yet. However, all the users at AFIT are impressed with the language,

which is relatively new. It promises to grow by being introduced at other installations.

d. Flexibility. Pritsker relates flexibility to the degree to which the language supports different modeling concepts. SLAM is strong in this respect with its combined three-world view of modeling. This project uses a combined process-discrete event view.

e. Processing Considerations. Interactive use will not be fully realized. A significant amount of main memory must be pre-allocated. However, statistics gathering is strong with built-in characteristics. Also, user defined statistics collection can be added as can user-tailored reports to the standard reports.

f. Debugging and Reliability. Combined processes may be difficult to debug. However that is a risk that must be accepted and exists to some degree in any implementation. System reliability of the Cyber and Harris may be a greater problem than SLAM system reliability.

g. Run-time Considerations. Execution speed may be a concern if the user loses control of user-specified limitations. This should be controllable in development.

Summary

There are several methodologies used in computer network design; intuition, experience, analytical modeling, and simulation. When used alone or in conjunction with another, the methodology(ies) used must produce some analysis of performance metrics to aid the designer. One tool used often is simulation

via a program running on a stand-alone computer. It is a flexible, powerful, and well-used design methodology. This thesis project consists of the development of such a program that has broad application. The logical design of that program is the subject of the next chapter.

III. Simulation Program Design

This chapter provides a graphical view of the logical design of the simulation program. This presentation is at a relatively high level of abstraction. Thus, no implementation considerations are evident. Additionally, the generic nature of this project can be seen in two levels of abstraction as shown in the following pages. That is, the entry points of the protocols can be identified at these levels. Any further decomposition would require specifying the protocol itself. That would then be the point where the generic nature of the design is lost.

Thus, this chapter provides a reference point for the entry or interface of the network protocols in the overall design. The detailed discussion of what can be inserted at these interfaces is provided in the next chapter. Before presentation of this design, a review of the design tool that was used will be given.

Structured Analysis and Design Technique

SADT (16) is a top-down diagramming technique which can be used in several ways. It can be used as a way of thinking, a procedure for development and review, and as a format for design presentation. It shows the components of a system, the relationships between them and how they fit into the hierarchical structure of the design. One view offered by SADT is that of data flowing through activities that act on the data. This provides a good mapping of the messages or packets flowing

through the network which acts on them. This view is the SADT activity diagram as shown in Figure 4.

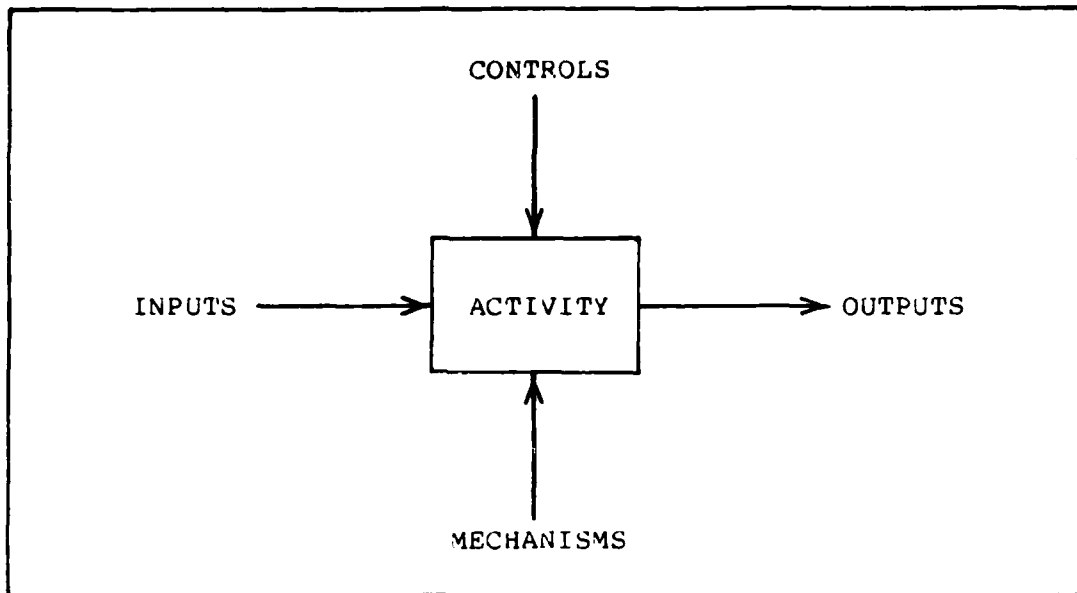


Figure 4. SADT Activity Diagram.

The box represents an activity. The activity uses mechanisms to convert inputs to outputs subject to constraints or controls. The entry or exit of each of the four types of arrows is always as shown in the diagram, i.e. inputs on the left, outputs on the right, etc. In each case, the mechanisms are either nodes, circuits, or hosts which input packets or messages and transforms them (constrained by the controls) to produce a packet or message in another form.

Each box that is broken down into greater detail (decomposed), is a parent. The decomposition of that box is a child diagram. The child diagram will show the parent activity in several more-detailed boxes, and will have all the respective

arrows of the parent. There will occasionally be some additional arrows on the child which are enclosed in parentheses to show that they are not present on the parent.

In the figures to follow, note that diagram A-O is the parent of AO. Also, each numbered box (X) of AO has a decomposition diagram (AX).

Program Design

The overall or context activity diagram Figure 5 is the A-O actigram. It shows the overall activity of the simulation system. Note that there is no input shown. That is because the host processes define the inputs which are generated internally. There is no corresponding arrow for analysis on the other activity diagrams because this is performed by SLAM, the language used for the implementation, and is transparent to the design.

The AO actigram, Figure 6, is the first decomposition of the A-O actigram. It represents the entire system but is broken into greater detail. It is decomposed activity by activity in Figures 7 through 12. Note that the same arrows shown on a parent activity box are also shown on the child activity box unless a set of parenthesis, (), are used.

The detailed discussion of each of the controls, which are mostly protocols, is given in the next chapter. However, for the proper perspective on these actigrams, please view them from the perspective of the information flowing through the network. The presentation of the following paragraphs follows that general flow. A reading of these paragraphs should be

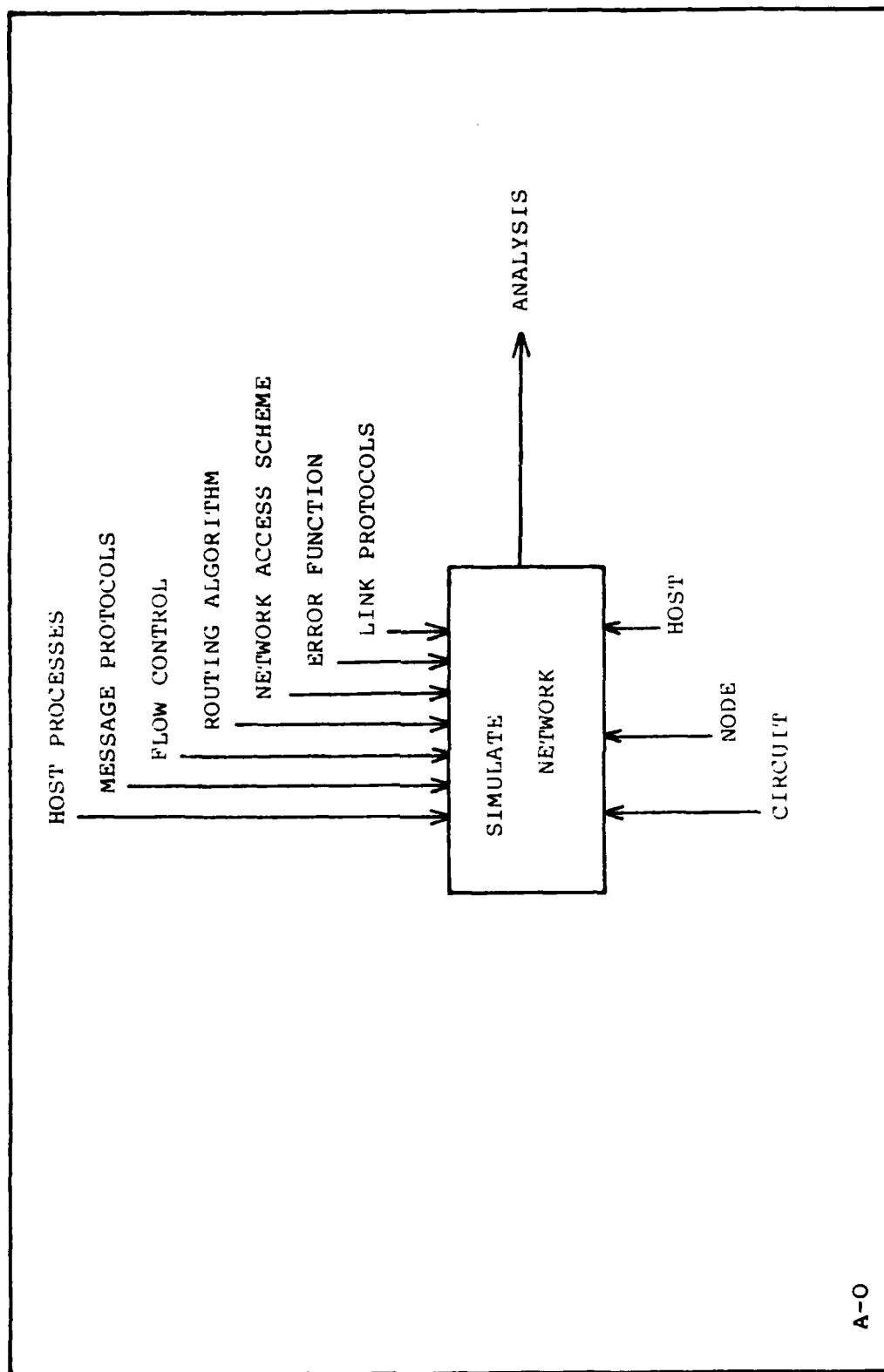


Figure 5. Computer Network Simulation.

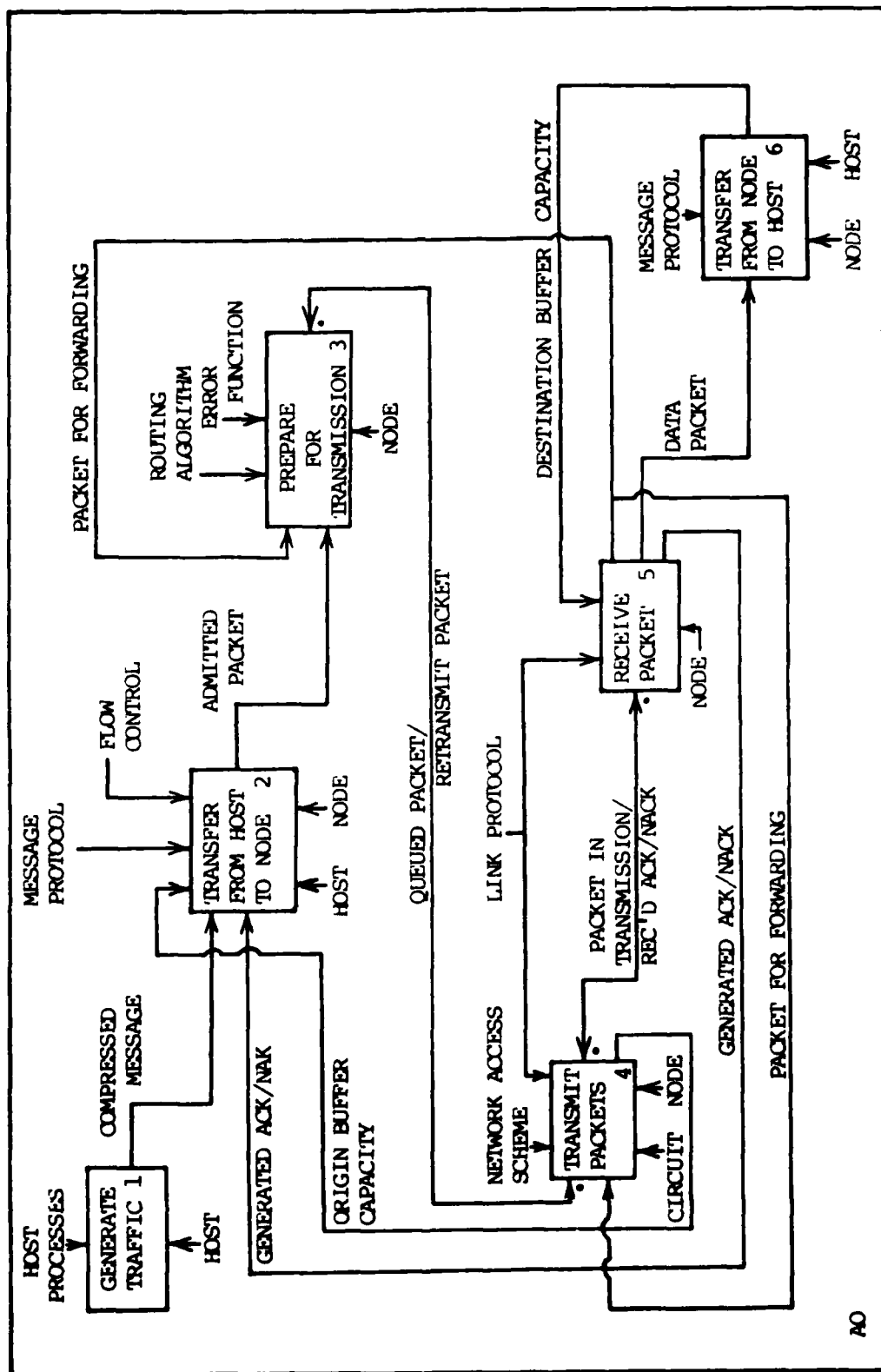


FIGURE 6. Simulate Network.

interrupted by periodic reference to Figure 6 to build and maintain an overall design perspective. For greater detail of any activity on Figure 6, refer to the decomposition diagrams (Figure 7 through 12).

Each numbered paragraph below refers to the respective activity number of Figure 6.

1. Messages are generated according to the host processes specified. Specifications include the interarrival time, the message length, the destination selection policy, the priority policy, and a data compression function.

2. The compressed messages enter the transfer from host to node activity. The host must first process the messages according to the message protocols. These specifications control the division of the message into packets, encryption, and the length of the packet header (overhead). Once the packet is completely prepared for entry into the node buffer, it must wait for admission to the node. Admission is controlled by the buffer capacity of the node and the flow control criteria specified. Once these constraints are satisfied the packet is admitted into the node and the next activity begins. Note also that a generated ACK/NACK input is also shown on this diagram. Even though this input is not from the host, it is included to model the control of node buffer resources on acknowledgements generated at the node itself.

3. The node prepares the packets for transmission. There are inputs to this activity; data packets from the host, and packets received from other network nodes that must be for-

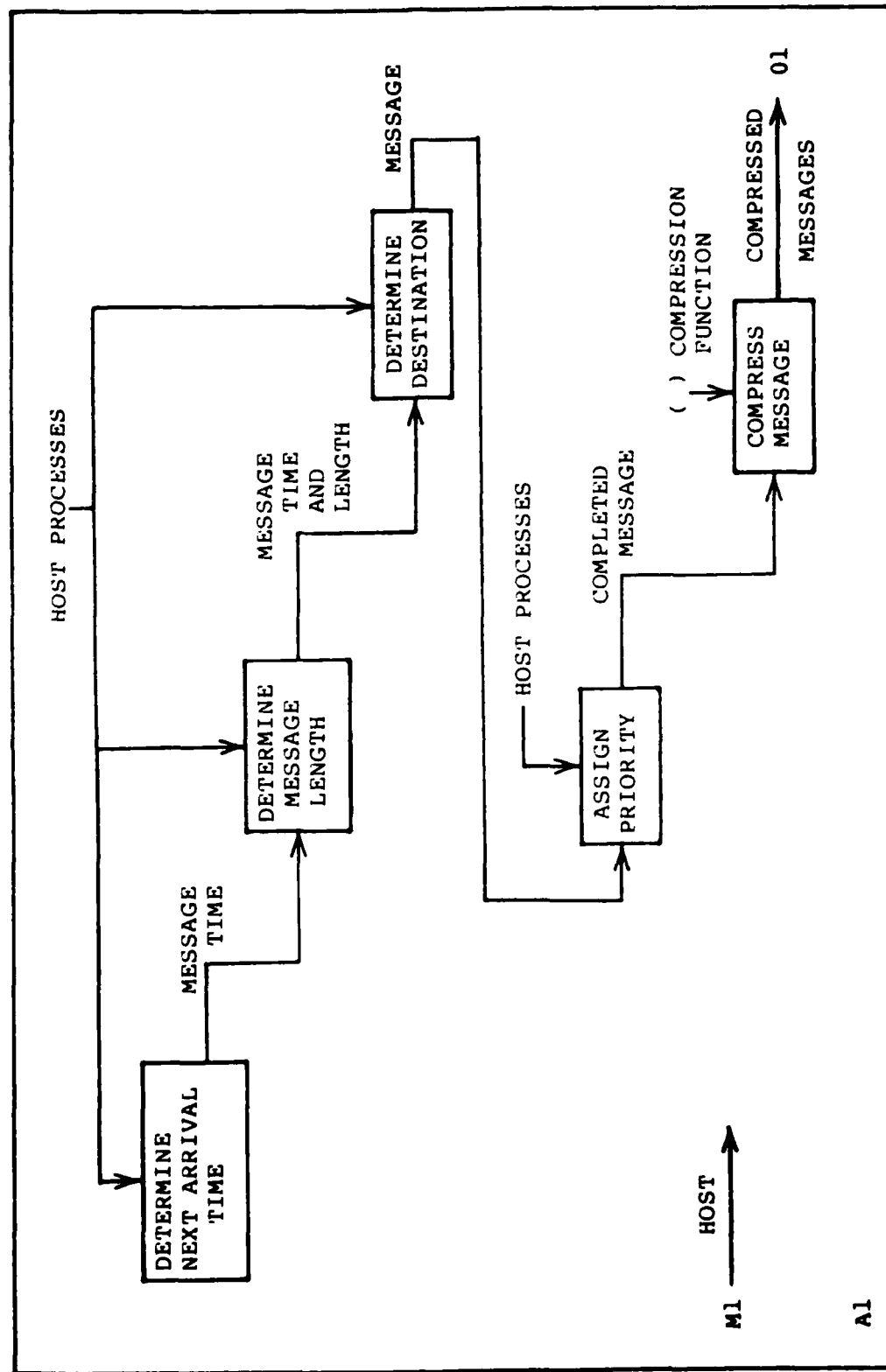


Figure 7. Generate Traffic.

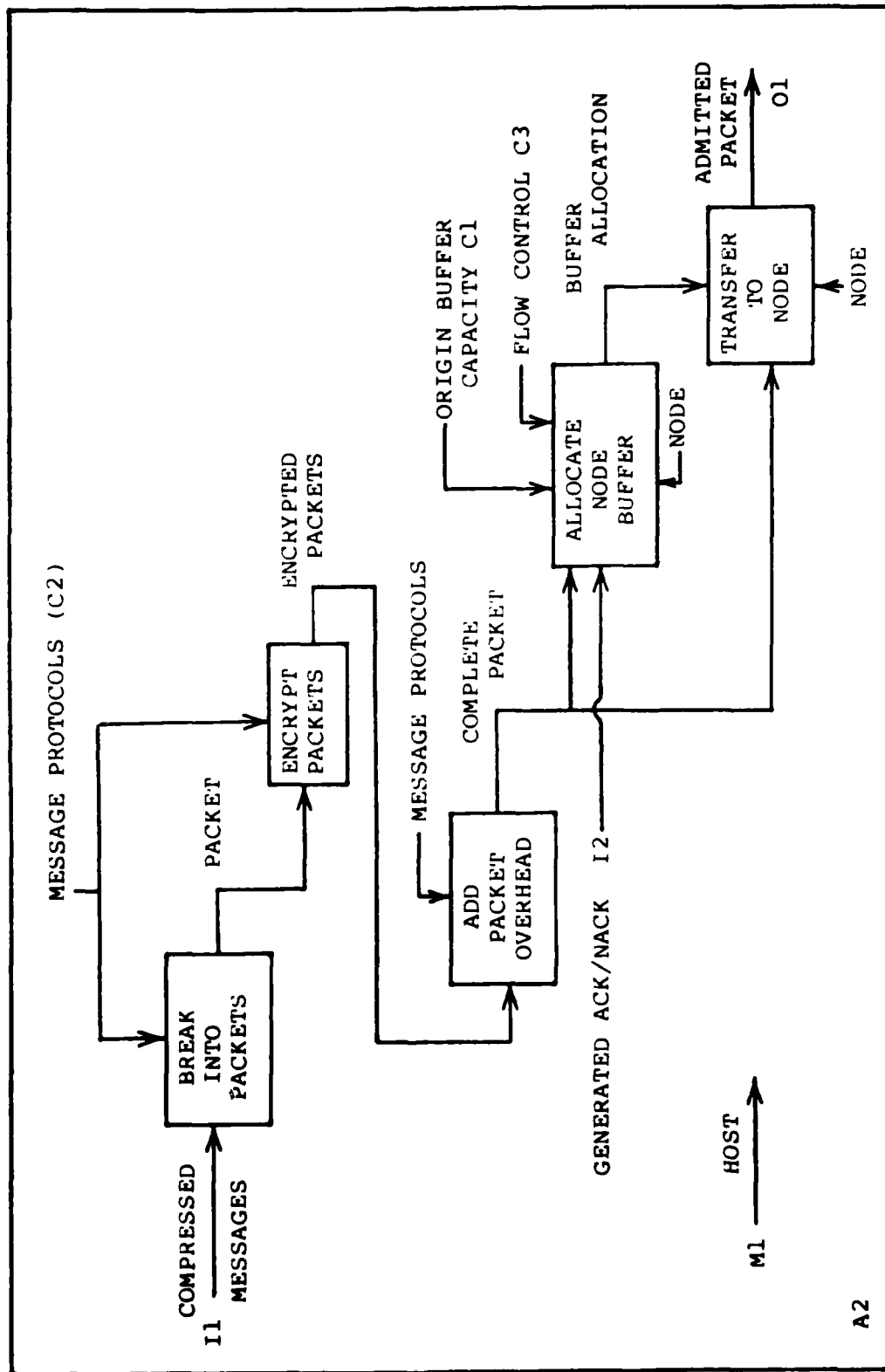


Figure 8. Transfer from Host to Node.

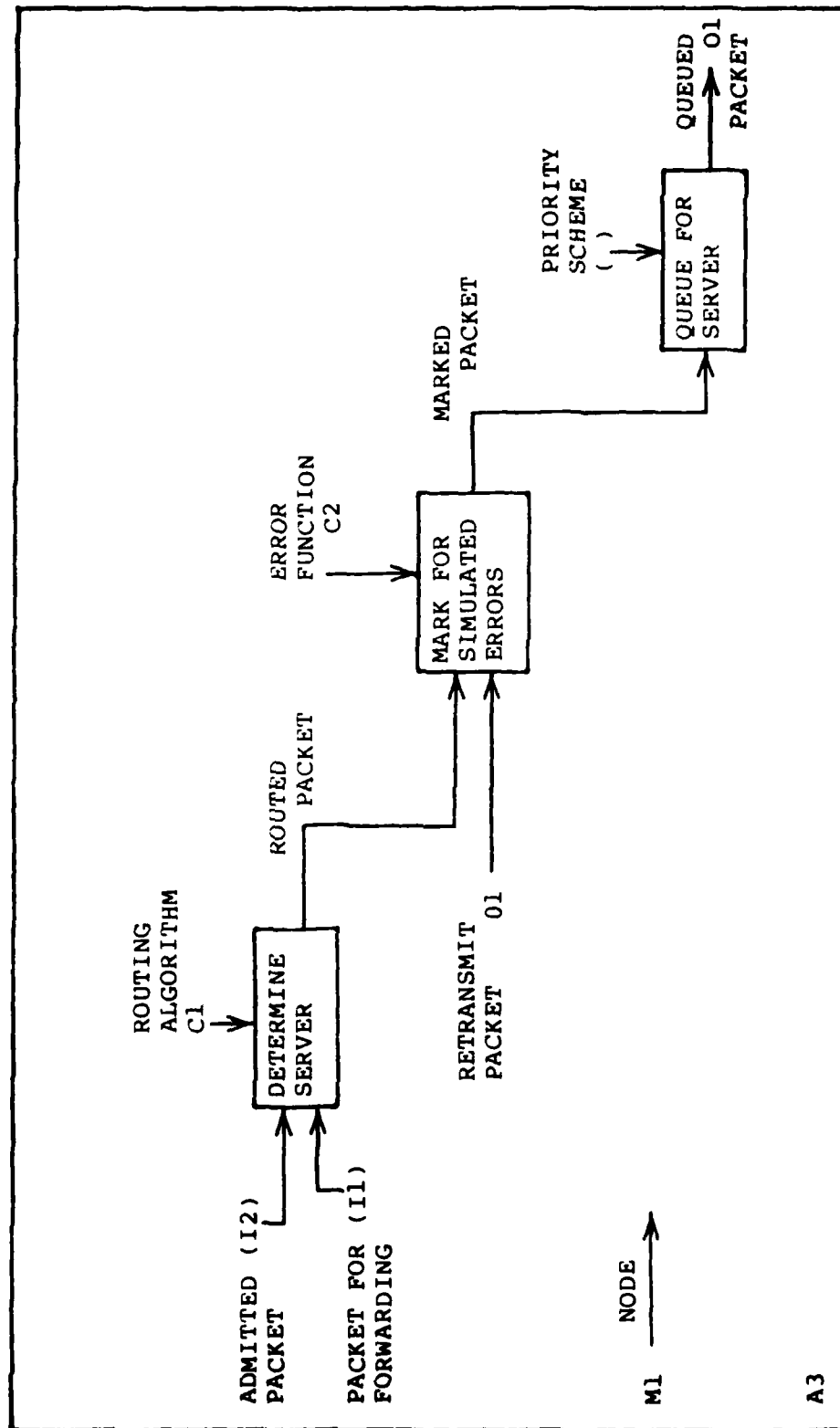


Figure 9. Prepare for Transmission.

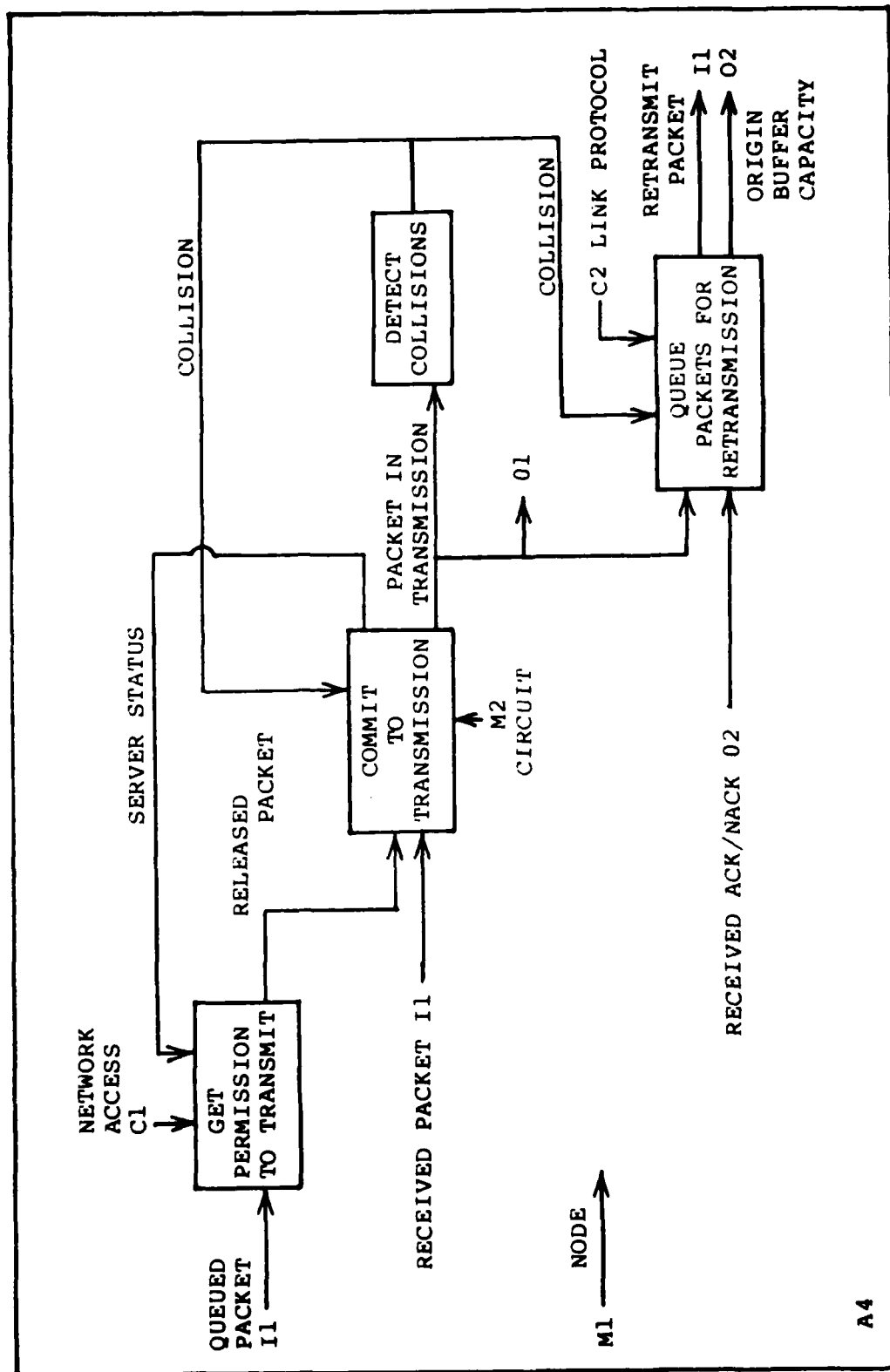


Figure 10. Transmit Packet.

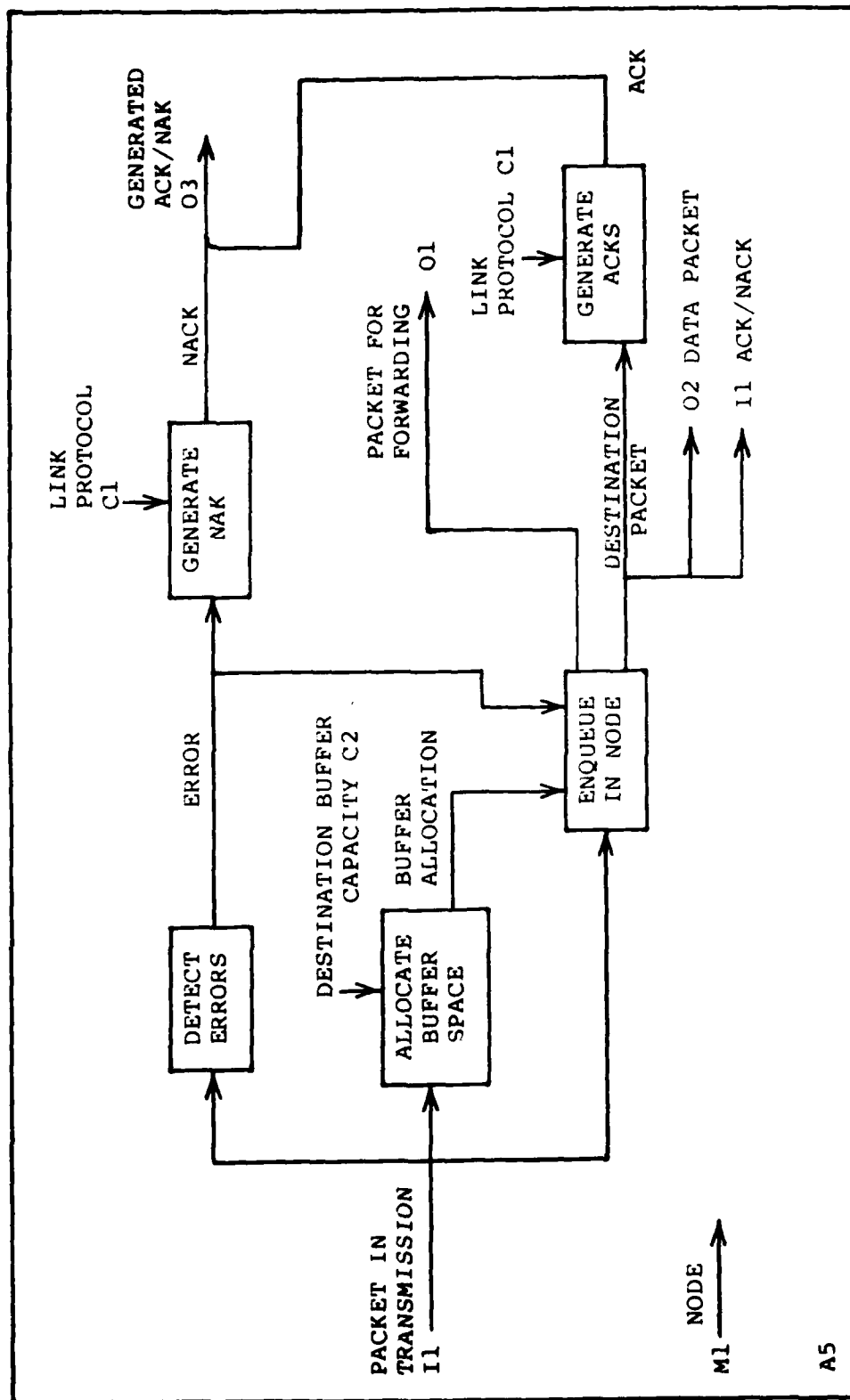


Figure 11. Receive packet.

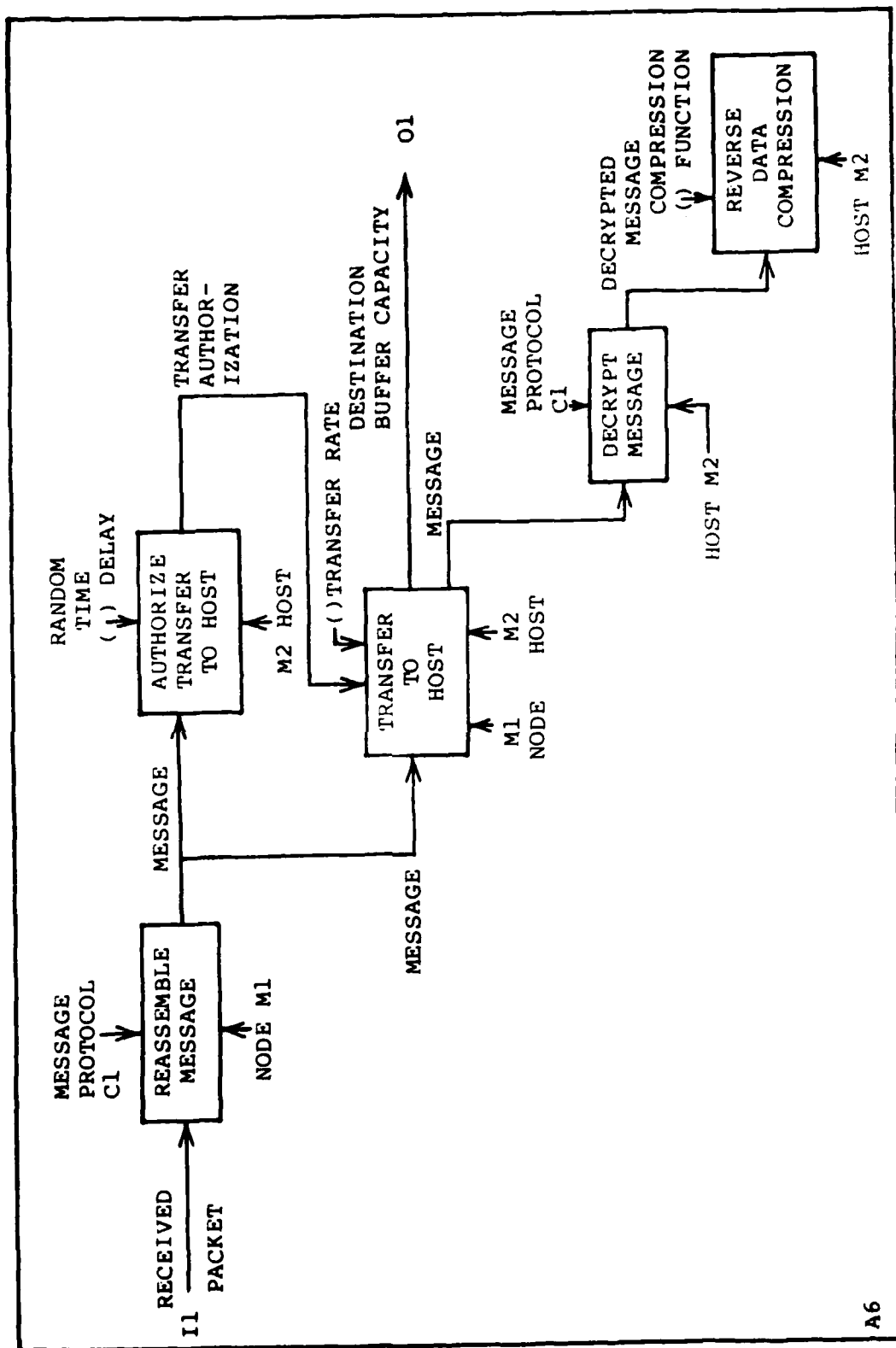


Figure 12. Transfer from Node to Host.

warded. An error function is used to randomly generate transmission errors in the packets and mark them for detection later. The last activity in preparation for transmission is queueing for a server.

4. Transmission is begun in the transmit packet activity. Transmission begins only after permission is granted. In the mesh network this is as simple as having an idle server. However, with other network access schemes there may be further constraints. The bus network, for example, may require waiting for a random time period. Also, the ring network may use a token for authorization to transmit. Note that packets for forwarding may be committed to transmission directly in activity 4 without being queued in activity 3. This is a special case using the ring network model in which packets are allowed to continue around the ring.

The collision detection activity that resides inside the transmit packet activity is also a special case. It is used only for bus networks. Once a collision is detected, the transmission stops and the node begins a waiting period defined by the network access scheme before attempting transmission again.

The remaining details of the retransmission process are defined by the link protocols. Packets are automatically retransmitted after a specified time-out period. They can also be released for retransmission upon receipt of a negative acknowledgement. A positive acknowledgement may allow the copy

of the packet to be discarded with the resultant change in buffer capacity.

5. The packets are received if there is sufficient buffer capacity in the destination node buffer. Additionally, a check for errors is made.

If an error is detected, the packet is discarded. If the link protocols specify negative acknowledgements, an error causes one to be generated. If packets are received without error, additional processing must occur. The data packets must be acknowledged according to the link protocols. The received acknowledgements must also be forwarded for retransmission processing. Lastly, data packets must be sent to the reassembly process which is the first activity in transfer from node to host activity.

6. The transfer from node to host activity represents all activities from reassembling messages to all the destination host processing. Once a message is complete, it is transferred to the host after a random time delay which can represent the time for an interrupt or a ready to receive signal. After the transfer is complete, the node buffer is released. The host decrypts the message and reverses the data compression that was performed at the origin.

Summary

Many computer networks can be modeled using the structures shown in the logical design of Figures 5 to 12. Messages are generated, packets are transferred to the node, the node processes packets, arbitrates and transmits packets, the

receiving node processes the packet and transfers reassembled messages to the destination host.

Inputs are not explicitly shown in the overall design because the entities (packets and messages), are generated internally. Controls to the activities give the model its structure. The controls are the implementation of the protocol options of the user's model. Outputs are mostly packets or messages in various states and places in the network. Circuits, nodes, and hosts are the mechanisms for network modeling.

The SADT representation of this generalized model remains at a relatively high level of abstraction. Some structures or controls come into play only when specified by the modeler.

A more precise definition of what this simulation program does is provided in Chapter V (discussion of the SLAM network structures) and in Appendix A. Appendix A contains structured English for each subroutine and function provided by the author. The Fortran code that Appendix A documents is part of a program and documentation package available from Maj Walter Seward, AFIT/EE faculty member.

The next chapter describes the specific protocol options available to the user. It also includes in the outline the parameters that must be specified.

IV. Program Specification

The high-level view of the simulation program offered by the previous chapter is a basis for the specific modeling options that are presented in this chapter. The SADT diagrams of Chapter III can be used to conceptualize the role and entry of the specific options that follow.

The user will find this chapter useful in accessing what can be modeled by the base-line program as provided by the author. Details are given on each base-line specification. After an overall review of this chapter, the user can use the quick-reference list of options of Appendix B to build a target model option-by-option. This chapter and Appendix B also show where to insert user-defined modeling specifications.

There is a limit to the number of options that could be designed into the program without an unacceptable burden of complexity or overhead. More options could have been included for manipulation by the user via global variables. However, the development of such options was stopped at a point where many global variables were already in use.

At the present point of development, many user selectable options, the entry of user functions, and the power of SLAM are available to the user. As a result, there is significant power available to the user.

A discussion of user visibility is given first to establish the framework for the discussion of options. Also given to help build that framework are the user processes of SLAM

network building, discrete event file manipulation, and input file preparation. Finally, the discussion of options follows in three main groups: global, host, and node options.

User Visibility

The user visibility of this program varies from a black box view to a white view depending on user requirements. If the user can satisfy all modeling requirements with designed-in options, the program retains its black-box characteristics. The user that requires special functions or additional modeling requirements may have to adopt a grey or white view. The type of view is reflected in the number and type of actions that the user must take in model building. All models require the user to replicate generalized SLAM network structures (Chapter V), and select values for global variables. These actions are described in the next section on network building.

The view becomes white box if the user must create new user functions, rewrite discrete events subroutines, or add new subroutines. The section on discrete event file manipulation describes these processes.

Most users may be able to model their networks by building the SLAM network, selecting global variable values, and manipulating a few existing user functions.

Of course, there will be applications that will not be amenable to the logical design of this program. In that case, it may be possible to modify the program by taking a white box view. That case would involve major rewriting, replacing, or adding procedures to the discrete event file. However, there is

always great risk in attempting that unless the design of this program and the desired program are both clearly understood. The base-line program may be a valuable reference in that case.

The user may develop a simulation program of his model by starting with the generalized SLAM network structures and the Fortran discrete-event file. Each must be handled separately but linked carefully together. Figure 13 gives a pseudo-SADT representation of the process.

Network Building

The user's SLAM network can be based entirely on the generalized network structures created by the author. This process is very straightforward and could be automated with a program development program as mentioned earlier.

The user must determine the number of nodes, number of hosts for each node, and the lines between nodes. Each structure of the target model must then be represented by a copy of the corresponding SLAM structure that is fully illustrated in Chapter V. For example, each host has the same structure except for certain indexes and labels. These indexes and labels must be unique for each host and identify its parent node. Similarly, each node must be represented by the same structures but with indexes and labels that are unique and identify it as node 1 to node 9.

There are several interfaces between the SLAM network and the discrete event file which require branching to the appropriate SLAM network structure upon entering the SLAM network. These branching structures are also very straight forward. They

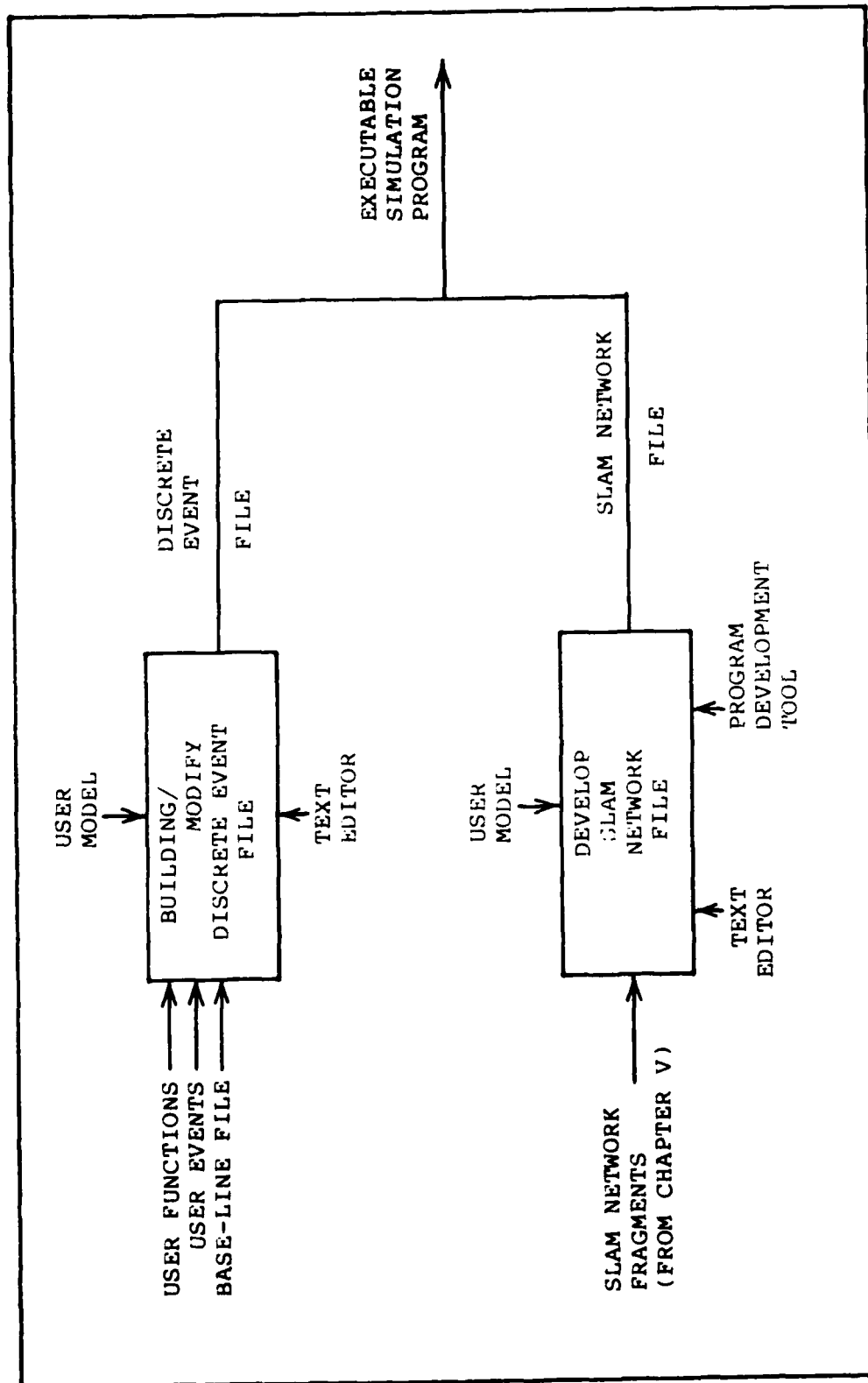


Figure 13. User Model Building.

must include the same number of branches as the number of network structures they branch to. For example, the branching structure to the server structures requires a statement corresponding to each server in the model.

Several other network structures can be selected by the user for specific models. These are primarily the controller loops for ring and bus networks, and a statistics collection cycler. The loops need to be included only if desired in the target model. The statistics collection cycler can be optionally inserted to provide additional analysis.

Perhaps the easiest part of the network to generate is the SLAM INTLC statements. These statements initialize the global SLAM XX variables. The syntax and position of these statements are illustrated in a sample SLAM network listing included in the program and development package available from Major Seward. The choice of which SLAM XX variable to include is determined by the user directly from the specifications of this chapter or the quick reference list, Appendix B. The values, of course, are entirely at the discretion of the user. The values are, of course, constrained by the target model, and possibly by the limits of the SLAM environment. Models which generate or require large numbers of entities also demand a large allocation of space for the SLAM environment. The user should make careful estimates of the target model requirements and behavior or resort to trial and error.

The remaining tasks of network building consist of generating the SLAM control statements. Pritsker (11:552-556) should

be a sufficient reference for their use and manipulation. Additionally, a SLAM network is provided by the author in the program and documentation package which is available from Major Seward. That network should illustrate most of the control statements needed by most users.

Discrete Event File Manipulation

The discrete event file provided by the author is a large collection of subroutines. The comments about manipulating them must be preceded by a description of their hierarchial relationships.

The simulation system is based on Program MAIN which calls subroutine SLAM. In the combined network-discrete event orientation used in this project, subroutine SLAM is invisible. However, through SLAM, the overall simulation is driven by the entities that flow through the SLAM network. From the SLAM network, subroutine EVENT and function USERF may be called directly. Each of these types of calls involve an integer argument that specifies the specific event subroutine or the specific user function.

The EVENTS consist of several high-level generalized subroutines and others that are specific to one of the three network models. The generalized subroutines are used by all models and may call down several levels into other subroutines depending on global choices made by the user. The model specific EVENTS are often more cohesive and uncoupled entities in themselves. They generally make fewer calls into other

subroutines. Also included are routines for diagnostics and statistics collection.

The SLAM USERF function is a collection of indexed functions. They are generally simple functions which support the SLAM network. Approximately twenty of these functions are provided by the author in the Fortran listing that is part of the program and documentation package. They are used most often for returning activity times or values for assignment to entity attributes. There are also a few that return a decision to the SLAM network via a one or a zero. This last application is used when the logical conditions required cannot be coded in SLAM network statements.

If the user can model all requirements with the provided specifications, only a few user functions will need minor modification. However, the modeling requirements of the user may force more modifications to the file. Changes may be made through changes to or additions of user functions. Similarly, changes to or addition of EVENTS or their subroutines can be made. All these actions are done through a text editor.

For changes, the task of finding the appropriate location may be significant. A review of the documentation of subroutines in Appendix A and the following instructions for additions will help the user find his/her way through the file.

User functions are implemented through SLAM USERF(IFN). For every integer index used as IFN, there is a jump to the label for the corresponding function. So, to add a user function, a label must be used at the start of the code of the

added function. Also, the label must be added to the USERF GOTO statement.

Events can be added to the simulation system similarly. Subroutine EVENT(I), is the SLAM mechanism for transferring control from the network EVENT node(I) to the actual Fortran subroutine corresponding to (I). Subroutine EVENT is structured as USERF(IFN) but each event is not in EVENT, but called from EVENT. Thus, for additional events that must be modeled, the event node (I) can be inserted in the network, and the Fortran code for I can be invoked through EVENT.

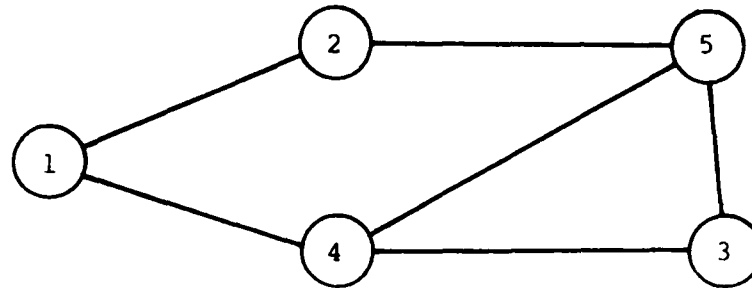
Major rewriting is the last means of changing the discrete event file. It is the most difficult and hopefully, least required. It should only be done if the user model cannot be accommodated by the structures provided. The user must be comfortable with the design and flow of the base-line program as well as the SLAM language. A thorough understanding of SLAM file manipulation in general, as well as this specific implementation, is essential prior to any redesign.

The author would advise users needing major design changes to use this program as a reference, and build a new model step-by-step.

Input File Preparation

This file is used to provide the simulator with two types of data; connectivity data for mesh models only, and node-to-node distances for all models.

The connectivity of the mesh models must be provided by giving the adjacent node numbers for each node. Figure 14.a



A. EXAMPLE NETWORK

2,4,0,0,0*

1,5,0,0,0*

4,5,0,0,0*

1,3,5,0,0*

2,3,4,0,0*

0,3,3,3,3**

3,0,3,3,3**

3,3,0,3,3**

3,3,3,0,3**

3,3,3,3,0**

* line not required for
non-MESH models

** line not required if
propagation delay is
specified as zero

B. INPUT FILE FOR A. ABOVE

Figure 14. Example Network and Input File.

represents an arbitrary five node mesh network for this example. The corresponding input file is Figure 14.b. The first five lines of Figure 14.b correspond to the N (five) nodes of the network. Line one indicates that node one is connected to nodes two and four. Note that each line must be filled with zeros to total the maximum number of lines possible in the simulation system, five. For non-mesh models, the lines containing destination nodes are not required.

The node-to-node distances are given next in an N by N array. A line for each of the N nodes must provide N entries unless the network propagation delay (XX(16)) is initialized to zero. If the user has thus indicated that there is no propagation delay for his model, no lines of distances are read or required by the simulator.

Note, however, that the units of distance must be compatible with the propagation delay. The models used to demonstrate this simulation system use kilometers and kilometers/msec.

A description of demonstration model development is given in Appendix C. The development of each model in Appendix C follows the outline of the next section.

Discussion of Options

All of the options in the following discussion are available to the user in the base-line program. The objective of the base-line is to provide a point of origin for the user to observe and then build upon. The modeling options that are included are either commonly used or easily recognized. They

were selected so that the user can easily conceptualize what is being done and provide a strong point of departure for subsequent additions and improvements.

Several observations should be made prior to enumeration of the options.

1. The normal SLAM filing system constraints put an upper limit of 100 on the number of files available. As a result, the network has been designed to have a maximum of 9 nodes, a maximum total number of 20 lines and a maximum total number of 29 hosts.

2. The base-line model uses concentration or statistical multiplexing. That is a current point of interest in the literature and for the author. For other multiplexing schemes, this model may be effective only for analysis at a high level. For some users models, with different multiplexing schemes, the multiplexing scheme of this simulation system may be transparent. The validation or proof of that is left to the user.

Note that concentration may be an awkward concept for the bus network. The node that is connected to the bus is a concentrator for the hosts if there is more than one host, and the node concentrates messages from other nodes to the host(s). However, concentration in the direction toward the hosts may be meaningful only if the host-node transfer rate is slower than the transmission rate on the bus.

3. Regardless of multiplexing scheme or other options, SLAM offers the specification of multiple identical servers for each activity. The user may take advantage of that for circuits

with many channels or other modeling characteristics. Although not used by the author, multiple parallel identical servers can be specified for any activity in SLAM.

4. Note lastly, the author has referred to communications between host and node as transfer, and communication between nodes as transmission. That convention is intended to avoid ambiguities and enhance conceptualization.

Once again, the objective of the base-line simulation package is to provide the base for user modeling. For further addition of options or capabilities, it provides a point of origin to build upon.

The logical flow of the program was shown in Figures 5 to 12. The control arrows of those figures can be conceived as the entry point of the options provided. Reference to the figures may be helpful while reviewing the Global, Node, and Host options. The reader should be warned that the enumeration that follows is detailed and lengthy. Readers generally familiar with computer networks may be able to recognize the options directly from the quick reference list of Appendix B. If not, the outline of Appendix B coincides with the outline to follow for cross referencing.

A. Global Options. Several specifications are dependent on the general network topology selected. Table 1 illustrates which are meaningful choices for the remaining global options.

1. Topology. The user must specify distributed, ring or bus (XX(33)=1,2,or 3 respectively). Once that choice is made,

	Topology Choice		
Specifications	MESH	RING	BUS
Concentration	A	A	A
Routing	SP	N/A	N/A
Node Latency	N/A	SP	N/A
Network Access	N/A	SP	SP
CSMA/CD	N/A	N/A	SP
Slot Length	N/A	N/A	SP
Max Prop Delay	N/A	N/A	SP
Min Packet Spacing	N/A	N/A	SP
Message Protocols	SP-all	SP-all	SP-all
Link Protocols	SP-(below)	SP-(below)	SP-(below)
Negative ACK	SP	N/A	SP
Retransmit T-out	SP	SP	SP
Hold T-out	SP	N/A	N/A
Immed. Error	SP	N/A	N/A
Node Queueing Max	N/A	SP	SP

LEGEND: A - always present in simulation system
 SP - must be specified
 N/A - not applicable to topology choice

Table 1. Specifications Based on Topology Choice

several other choices must be made based on the topology choice. The distributed network must be assigned the routing choice (paragraph 2 below). The bus and ring networks must each be assigned a network access scheme and several other specifications (paragraph 3 and 4 below).

2. Routing. (XX(53) and XX(54)) The actual algorithms used are implemented automatically. The choice required here is either fixed or adaptive routing in the distributed network.

The distributed network routing algorithm is a shortest path algorithm based on Schwartz (15:237-243). Structured English for the algorithm is given in Appendix A. An illustration of the specific data structures used are shown graphically in Figure 15. The fixed routing scheme is implemented in the SLAM INTLC subroutine by executing the algorithm for each node. The weight table, in this case is initialized to one's to give essentially a minimum-hop result. In all cases, the result is stored in the NEXTIN array.

A SLAM network cyler is provided to call subroutine ROUTER which implements a modest adaptive routine scheme. This scheme is based on weighting the lines to adjacent nodes according to the server queue lengths to the respective lines. When called periodically from the SLAM network, subroutine ROUTER updates the weight table for each node by the queue lengths times a factor given by SLAM XX(53). The minimum path algorithm (executed by subroutine MINPAT4) is then used to update the routing decisions. The time period between calls to subroutine ROUTER is given by SLAM XX(54). Specification of

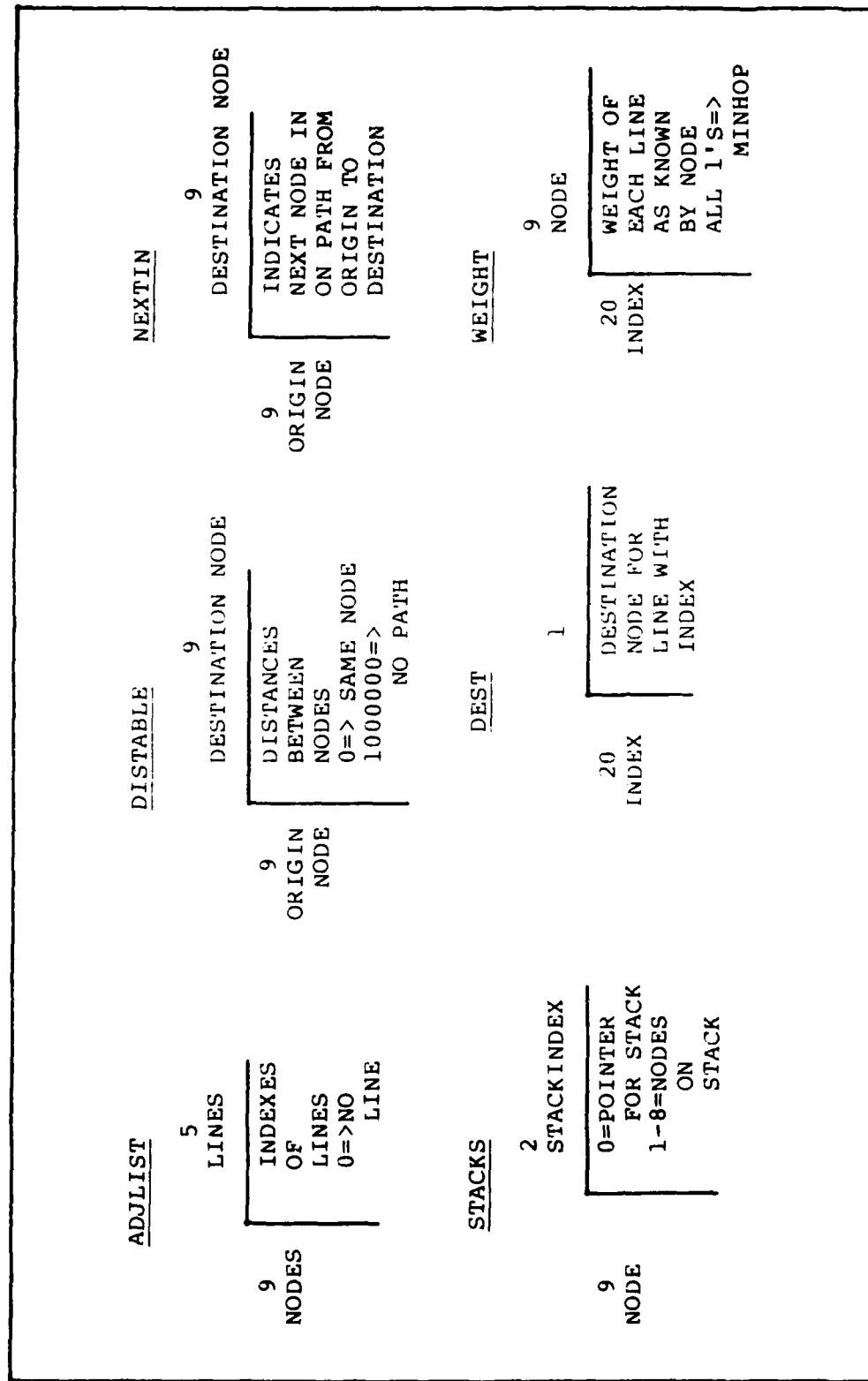


Figure 15. Distributed Network Routing Algorithm Data Structures.

XX(53)=0 is necessary to disable this adaptive routing scheme. The resulting routing scheme is fixed minimum-hop routing.

The above scheme is based on information local to each node, its own queue lengths. More elaborate schemes could be implemented with only two significant additions. A SLAM network clock could periodically cause each node to create "control" packets which would carry weight updates for neighbor nodes. The control packet would have to be assigned a new packet type for processing in subroutine RECVPKT. That processing would be no more than updating the weight table of the receiving node and executing the minimum path algorithm. However, the simulation system would be burdened with the requirement for more entity attributes and thus overhead.

3. Ring Options. Each of the following must be specified for ring networks only.

a. Node latency (XX(18)). The delay experienced at each node must be specified in data units such as bits.

b. Network Access Scheme (XX(38)). This can be thought of as the right of the server to become active. This option doesn't apply to MESH networks in this system because the server may begin transmitting a packet as soon as the server is idle. The ring networks and bus networks have a great variety of network access schemes. This simulation system provides two options for each.

1. Ring

a. Token (XX(38)=1). This scheme requires that the server be in possession of a "token" before transmit-

ting packets that originate at that node. Packets that are already circulating around the ring may continue. Once packets reach their destination, they still continue in their entirety around the ring as ring acknowledgements until they reach their origin. At that point, if no errors were detected, the packet copy in the retransmission queue is discarded. If errors exist, the packet will be retransmitted. This scheme is similar to many token rings and models the Newhall loop described by Jafari (9).

b. Slotted (XX(38)=2). The communication space (in time) is divided into fixed time slots. Each slot is long enough to transit one full data packet. Each node may place a data packet that it originates in the next time slot available for transmission. As in the token scheme, packets already circulating in a slot may continue to travel around the ring. This scheme is typical of slotted rings and models the Pierce loop also described by Jafari (9).

4. Bus Options. Each of the following must be specified for bus networks only.

a. CSMA/CD (XX(51)). The Carrier-Sense-Multiple-Access/Collision Detection protocol is selected with XX(51)=1, otherwise assign XX(51)=0.

b. Network Access Scheme. There are two basic schemes available in the base-line program. First, a bus with an arbitration scheme based on a random delay is selected with XX(47)=1. In this model, a node which has a packet ready for transmission may immediately check the bus and transmit without

delay. If the bus is busy, the node goes into a waiting period until its next try. The waiting period for this option is generated in USERF(16).

The second bus model available uses a slotted scheme with (XX(47)=2). In this model, a slot is defined by the user as the approximate packet transmission time used in the network. In this model, the channel time is broken up into segments (slots) which are the length of a packet transmission time. Each node may begin transmission only at the beginning of the slot. this can be used to model networks similar to slotted ALOHA (6:162-164).

The slotted bus is implemented so that nodes begin transmission at the same time. To make this model work the input file node to node distances may need to be manipulated somewhat. If packets must be timed to arrive at nodes in the same slot time regardless of origin, then the node to node distances must all be equal. In that way, the integrity of the slotted system is kept intact. Otherwise, a very elaborate timing scheme for beginning transmission times must be developed to get the advantages of the slotted system.

1. Slot length (XX(50)). This must be specified for slotted models using time units consistent with other specifications.

2. Maximum propagation delay on the bus (XX(52)).

This is specified to aid the Fortran subroutines in some decision making for efficiency.

3. Minimum packet spacing (XX(48)). This must be specified for bus models using a random delay scheme. It is the minimum interval of time between the end of transmission of a packet and the beginning of the next packet transmission.

5. Message Protocols. These are options designed to allow the user to define the size and format of messages and packets.

a. Message lengths are generated in USERF(1 or 7). The call to USERF for message length is currently made from subroutine MAKPKTS. Both of these user functions use an exponential distribution with mean given by XX(25) to return the message length.

b. Maximum message length is specified by XX(21). It is implemented only in USERF(7). (USERF(1) returns message lengths with no upper bound.)

c. Packet Length. This is normally a constant upper bound and is given by XX(23). It is used as a maximum data length of packets before the compression function is applied and before the packet overhead is added to the packet.

d. Packet Overhead. This is intended to lump together all header, trailer, and other overhead information in each packet. It is a constant defined by XX(24). It is also used for the total packet length of acknowledgements (excluding ring acknowledgements).

e. Compression Function. A compression function is applied to the data portion of the packets at the originating

host. It is reversed at the destination host. The compression function is a percentage given by XX(29).

f. Message Arrival Process (XX(20)). A basic choice must be made concerning the nature of arrivals. A process using an interarrival rate is used when XX(20)=1. (Reference host option 1 to follow.) With XX(20)=0, message creations at a host are not allowed until the previous message is completely admitted into the corresponding network node. (Reference host option 2 to follow.)

6. Link Protocols. The link protocols are given to provide several mechanisms for dealing with errors in the transmission process. All of the following options are implemented with SLAM XX variables initialized by INFLC statements.

a. Negative Acknowledgements (XX(31)). These are simply used or not used depending on the user's specification. XX(31) must be initialized to 1 to use this option, 0 otherwise. Not applicable to ring networks.

b. Retransmission time-out.

1. Most models specify this time with XX(14). It is the amount of time that the node holds a copy of a packet without receiving a node-to-node acknowledgement. If an acknowledgement is received before the time-out, the packet is removed. If no acknowledgement has been received and the time-out occurs, the packet is requeued for transmission.

2. Non-CSMA/CD models use a retransmission time from a uniform distribution. The retransmission time generated for these models is used by the SLAM network the same way as in

other models. The distribution is defined by XX(56) and XX(55) which are used as the high and low ends of a uniform distribution respectively.

c. The hold time-out (XX(13)) is used only in distributed topologies. It is similar to the retransmission time out except that it applies to packets that have been removed from the retransmit queue and placed in the hold queue. The hold time-out is usually longer than the retransmit time-out because an end to end acknowledgement is required to remove packets from the hold queue. If an end-to-end acknowledgement is received, the copy on hold is discarded, otherwise, it is retransmitted when hold time-out occurs.

d. Immediate error detection (XX(32)). This option is meaningful in distributed models that use negative acknowledgements. If immediate detection is specified, XX(32)=1, negative acknowledgements must also be specified. The immediate error detection option will cause the receiving node to immediately generate and queue for transmission a negative acknowledgment as soon as an error is detected.

7. Flow Control. This refers to the controls used to keep the network from overloading. This is done primarily by limiting inputs to a node and controlling the priority of packets. Inputs to a node can come from three sources: other network nodes, hosts at that particular node, and from the node itself, (acknowledgements and control packets). These flow control options are primarily used to control inputs from the host. Data units are not mentioned in the paragraphs to follow.

The user may use any data unit base as long as the data rates in other specification are consistent with them.

a. Minimum buffer level of data units for message creation (XX(36)). This option is not only a flow control feature but also a simulation program protection mechanism. It controls the number of entities in the system from an excessive source of message creations. In other words, it prevents a host message creation from occurring if the associated node buffer falls below the specified level. This option can be made nil by selecting XX(36)=0.

b. Maximum number of packets queued for entry to a node (XX(45)). This specification causes the denial of a message creation if there are an excessive number of packets already queued awaiting node entry. This specification is a simulation system protection feature as well as a modeling option. Lack of control in this area could lead to an explosion of packets in the SLAM filing system. This option can be made nil by choosing an unreasonably large number of packets.

c. Minimum buffer level required after packet admission (XX(37)). This is used as a condition for packet admission to a node from a host. If, after a packet is admitted, the buffer level would fall below this minimum, entry is denied and the packet is queued for entry. This option can be made nil by selecting XX(37)=0.

d. Maximum number of packets in a node for a given destination node (XX(46)). This specification combines with the previous specification to result in the overall condition

for packet admission to a node from a host. It is similar to a window size in virtual channel protocols. If there exists this maximum number of packets in the node which are routed to the same destination as the packet seeking admission, that packet will not be admitted. This option can be made nil by choosing an unreasonably large number of packets.

e. Priority scheme. This option is based on an integer "packet type" attribute that is assigned to each packet. The acknowledgements are assigned; negative=4, end to end=3, and node to node=2. Data packets are assigned 1. This option can be implemented by specifying a priority on the server files that represent queues for admission to a node. Highest value of packet-type first is implemented through a SLAM priority statement for each appropriate file. The SLAM network in the program and documentation package uses such priority statements. Absence of the priority statements results in a FIFO (no priority) scheme.

f. Node server queueing maximum (XX(44)). This specification is a limit on the number of packets that may be queued up for the node's server (ring and bus models only). A low number will reduce the number that will be queued, however, the number specified may still be exceeded in the current implementation. The requeueing of packets from the retransmit and hold queues is not constrained by this option.

8. Propagation delay of media (XX(16)). This is usually most significant in bus networks. It can also be used in the other models where its significance is heavily dependent on the

other specifications given by the user. Once again, it must be given in units consistent with the distances given in the input file and the rates given in the other specifications. For models that do not require any modeling of propagation delay, initialize XX(16) to zero.

9. Error Function (XX(30)). This is specified as the probability of a transmission error in a packet. The transmitting node uses that probability to randomly select packets that it will mark with an error. The receiving node will note the error, discard the packet and generate acknowledgements as specified in the line protocols. The ring network is a special case which continues to circulate the packet around the ring with or without errors.

10. Encryption rate (XX(10)). This is specified in the data unit/time unit base of the model and is used to generate a processing delay in the hosts at both ends.

B. Host Options. These options are the specifications that represent host processes. In the base-line they are actually handled in a global fashion. User tailoring is necessary to create special non-homogeneous host characteristics.

1. Message interarrival time. For models specifying XX(20)=1, this option is given by USERF(17 or 18). Which user function to be used must be specified in each SLAM network host structure. Additionally, XX(17) must be specified in either case. USERF(17) returns a constant interarrival time given by XX(17). USERF(18) returns an interarrival rate from a poisson process that uses XX(17) as the mean interarrival time.

2. Message regeneration control (XX(15 and 39)). For models specifying XX(20)=0, additional control must be provided in the form of delays.

Following node acceptance of the last data packet of a message, a delay given by XX(15) is given to prevent automatically filling the buffers of the network nodes. If a simulation run is desired with a maximum load, XX(15) should be initialized a small value. Such a run may require vast amounts of computer memory resource to hold all the SLAM entities that would be created. Inadequate flow control specifications would result in SLAM termination for inadequate file space.

Following suppression of a message creation, a delay prior to the next creation attempt can also be specified by XX(39). This is necessary to prevent an endless loop that could occur if a message is suppressed.

3. Host-node transfer rate (XX(11)). This is specified in data unit/time unit and is used to generate the transfer rate in both directions.

4. Node to host delay (XX(57)). This is intended to model a time delay such as an interrupt or ready signal that must be experienced before a host will accept a message from the node. This time is added to the node-host transfer time in USERF(12).

5. Destination distribution. In the current implementation, this time is randomly determined in subroutine MAKPKTS.

C. Node Options

1. Number of nodes (XX(26)). The number of nodes in the network must be specified in this SLAM XX variable as well as in the network building process by inserting the appropriate number of SLAM network structures.

2. Maximum number of hosts. The number of hosts is derived by the number of host SLAM network structures. No XX variable specification is necessary. This total number of hosts is limited by the SLAM files required for queueing messages awaiting transfer to the host from the node. A total of 29 are possible but that many have not been tested.

3. Maximum number of outgoing lines per node (XX(27)). This is a global specification of the maximum number of lines that any particular node may have. The maximum allowed for this specification is five. Each node may then have no more than this maximum number of lines specified. (Specifications greater than one are meaningful only for MESH models.)

4. Line specifications.

a. Transmission rate (XX(12)). This is given globally at this time in (data unit/time unit).

b. Physical length. This information must be matched with propagation delay to give a meaningful model. It must be provided in the input file prepared by the user which also provides destination nodes for each line.

c. Destination node. This is the receiving end of the line.

5. Buffer capacity of each node (XX(1-9) or XX(35)). The base-line provides homogeneous assignment of this resource via XX(35). For non-homogenous assignment, XX(1 through number of nodes) must be specified and subroutine INTLC must be modified.

Summary

This chapter should give the user an overall perspective from which to view this simulation system. The user should evaluate his/her modeling requirements with respect to the options given in this chapter. An appraisal of the usefulness of this simulation system for a particular application can then be made. If the user view is black box, then it certainly should be used. If the user modeling requirements force a white box view, a closer examination of the structure and implementation of this system should be made before proceeding. Further structural definition of the SLAM network portion of this simulation system is given in Chapter V.

AD-A127 318

DESIGN AND IMPLEMENTATION OF A GENERIC COMPUTER NETWORK
SIMULATION SYSTEM(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING

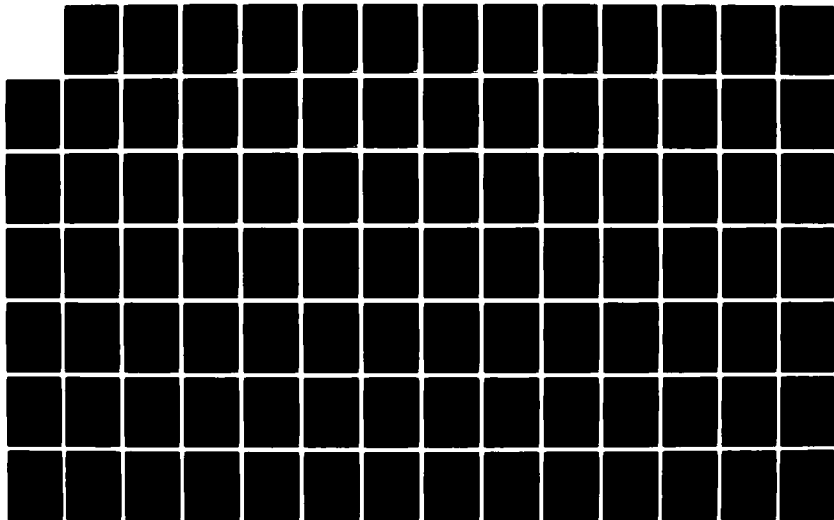
2/3

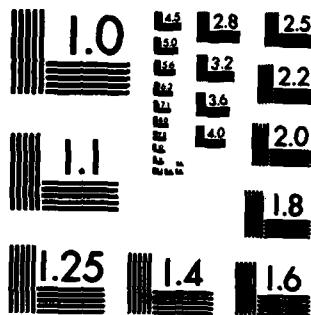
UNCLASSIFIED

S J FOSTER MAR 83 AFIT/GCS/EE/83M-2

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

V. Slam Network Structures

GENERAL

The SLAM network portion of this simulation program is a collection of largely disjoint networks. Each of these network fragments represent either a part of the overall network structure, a branching mechanism, or a timing loop. Examples of each may be noted in the figures to follow. For example, Figure 17 represents the structure for a host message generator. Figure 18 represents a branching structure to control the next message creations. Figure 30 represents the timing loop for the slotted bus controller.

Many figures include EVENT nodes. Each EVENT number corresponds to a Fortran subroutine which is noted for each EVENT node on each diagram. A detailed description of each subroutine is given in the Documentation of Appendix A. However, the appendix to Chapter V is a condensed documentation package of EVENT subroutines only. It provides a concise statement of the function of each EVENT subroutine for quick reference from Figures 17 through 31.

SEQUENCE

The order of presentation of these diagrams is given as sequentially as possible. The sequence generally follows the path of data from message creation, packet flow through the network, and message transfer to the destination host. Sequentiality is interrupted somewhat by the branching structures

that are included next to the diagrams that they branch to and from.

PROGRAM CONNECTIVITY AND SLAM NETWORK FLOW

Although the SLAM network fragments are disjoint, the simulation program is connected via the EVENT and ENTER nodes. The data path in the network usually stops at an EVENT node. At an EVENT node, control is transferred to the event and its associated subroutines which generally return control to the SLAM network via the ENTER nodes. However, as noticable in several diagrams, the control path may continue right through an EVENT node. In that case, entities continue to flow through the exit of the EVENT node to other network activities or nodes.

Figure 16 provides a simple introduction to some SLAM network nodes and activities. It is designed solely to illustrate the flow of entities through the network and not model anything specific. The following discussion describes the flow of entities through Figure 16.

Entities flow through the CREATE node whenever an entity enters it from somewhere else in the network. Additionally, as typical in this project, an entity is created at time zero as a seed to get things going. As soon as an entity arrives or is created it flows over activity 1. Activities 1-3,5 and 6 are regular activities and can carry an "infinite" number of entities.

By contrast, and looking ahead, activity 4 is a service activity. As typical in this project, it is defaulted to handle

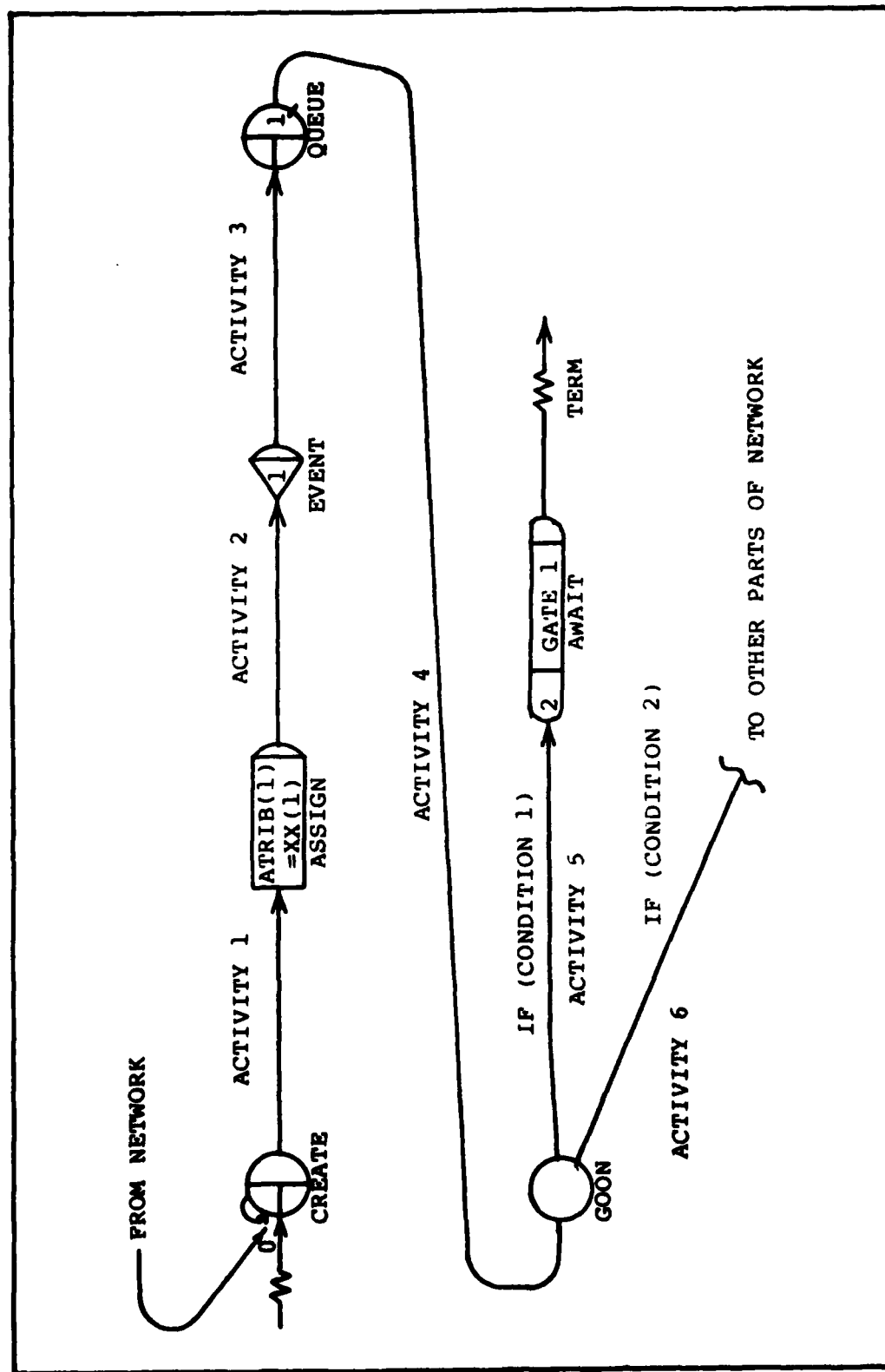


Figure 16. SLAM Network Introduction

only one entity at a time. Additional entities that flow into the QUEUE node are filed in file 1 until activity 4 is idle. When activity 4 is idled, entities in file 1 are automatically started on activity 4.

Returning to activity 1, entities flow into the ASSIGN node. Attribute one of that entity is assigned the current value of XX(1) and is committed to activity 2. When entities complete activity 2, EVENT 1 causes a Fortran subroutine to be executed with the SLAM ATRIB array assigned with the attributes of the entity that flowed into EVENT 1. Once the discrete event subroutine has completed execution, an entity with the SLAM ATRIB array is committed to activity 3.

Entities completing activity 4 enter the GOON node for branching. If condition 1 is true, the entity is started on activity 5. If condition 2 is true, a copy of the same entity is committed to activity 6.

Entities completing activity 5 enter the AWAIT node that represents GATE 1. This project operates GATES in a continuously closed state. Entities entering the AWAIT node are filed in file 2. If the gate is never opened, the only way for entities to leave file two is through discrete event subroutines. That is the mechanism used throughout this project.

The TERM node is shown to illustrate that if GATE 1 was opened, the entities in file 2 would be lost to the system.

The preceeding discussion is by no means exhaustive. It should, however, serve to aid users to follow the discussion of

Figures 17 through 31. Pritsker (11) should be consulted for greater detail and questions generated by the reader.

DIAGRAM DESCRIPTIONS

1. Host Message Generation (FIGURE 17). Each host-node pair requires this structure to represent the message generation process. An interesting feature of this structure is the conditional branch back to the next message creation when unrestricted message generation is used. If message generation is independent of the previous message handling, this branch is taken to give the message interarrival time. USERF() is unspecified in this general diagram to illustrate that any interarrival process or distribution can be specified easily in a user function. The call to USERF can also be replaced by a SLAM XX variable.

If message generation depends on node acceptance of the previous message, entities are entered into the network for branching back to the appropriate message generation process via Branching for Creating Next Message (Figure 18).

In all cases, the message entity is also committed to an activity flowing to an ASSIGN node for several attribute assignments to the message. Then, the message enters the EVENT node for MAKPKTS.

From this point in the discussion, reference to an entity may be referred to by the expression message or packet if the entity represents a message or packet. Entity will normally be used otherwise. Additionally, note that reference to an entity starting and stopping an activity may be referred to as flowing

EVENT 4 = MAKPTS
 X = HOST NUMBER
 Y = NODE NUMBER

FROM FIGURE 18

USERF ()

IF UNRESTRICTED

MESSAGE GENERATION

HXNY

ATTRIB(1) = TNOW
 ATTRIB(2) = Y
 ATTRIB(10) = Y
 ATTRIB(16) = X

EXY

Figure 17. Host Message Generation.

to the next node. The SLAM network rules of sequence allow the entity to flow from one node sequentially to the next without an activity specified in between. Thus the generality of description is enhanced.

2. Branching for Creating Next Message (Figure 18). This structure is used only when message creation depends on the previous message's acceptance into the network node. The normal entry point is ENTER node 8. Entities arrive there from subroutines CKADMT and ADBRANH when a packet has been accepted into a node. An activity providing a user-specified delay is encountered prior to arrival at GOON node RECR. If the packet was the last data packet of the previous message, then the appropriate branching is taken starting at GOON node CRCK to the same originating host creation process. The remaining branching consists of GOON nodes and activities that jump to the appropriate labels back to the originating host creation process.

ENTER node 9 is used in a special case. If the flow control constraints are exercised in subroutine MAKPKT to suppress a message when the corresponding node buffer falls below a specified level, then ENTER node 9 is used. Similiar to the above case, a user-specified delay is encountered by entities flowing over the activity to GOON node CROK.

3. Processing for Entry to Node (Figure 19). Packets enter this structure from subroutines MAKPKTS, GNEGNAK, and GENACKS.

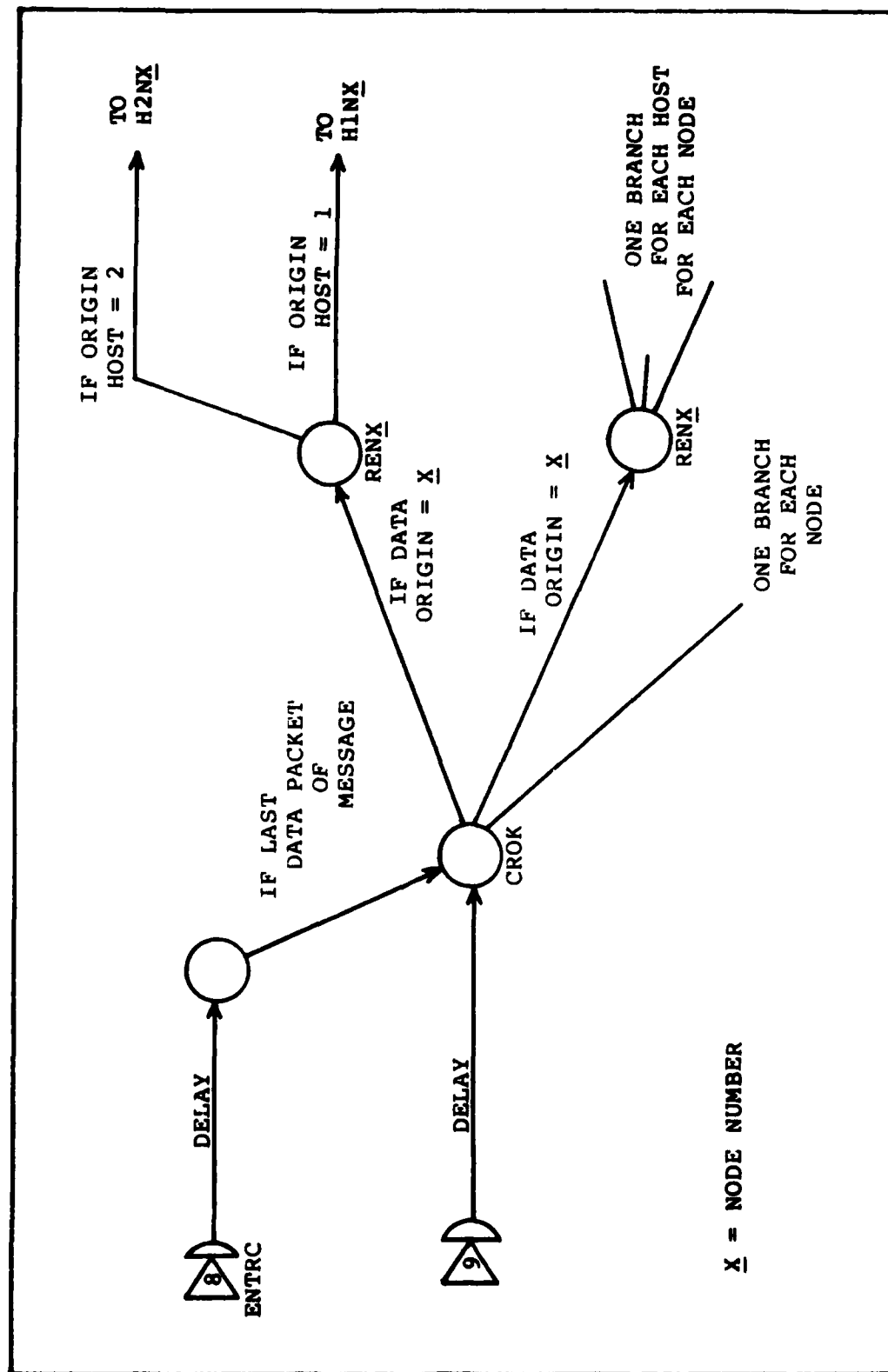


FIGURE 18. Branching For Creating Next Message.

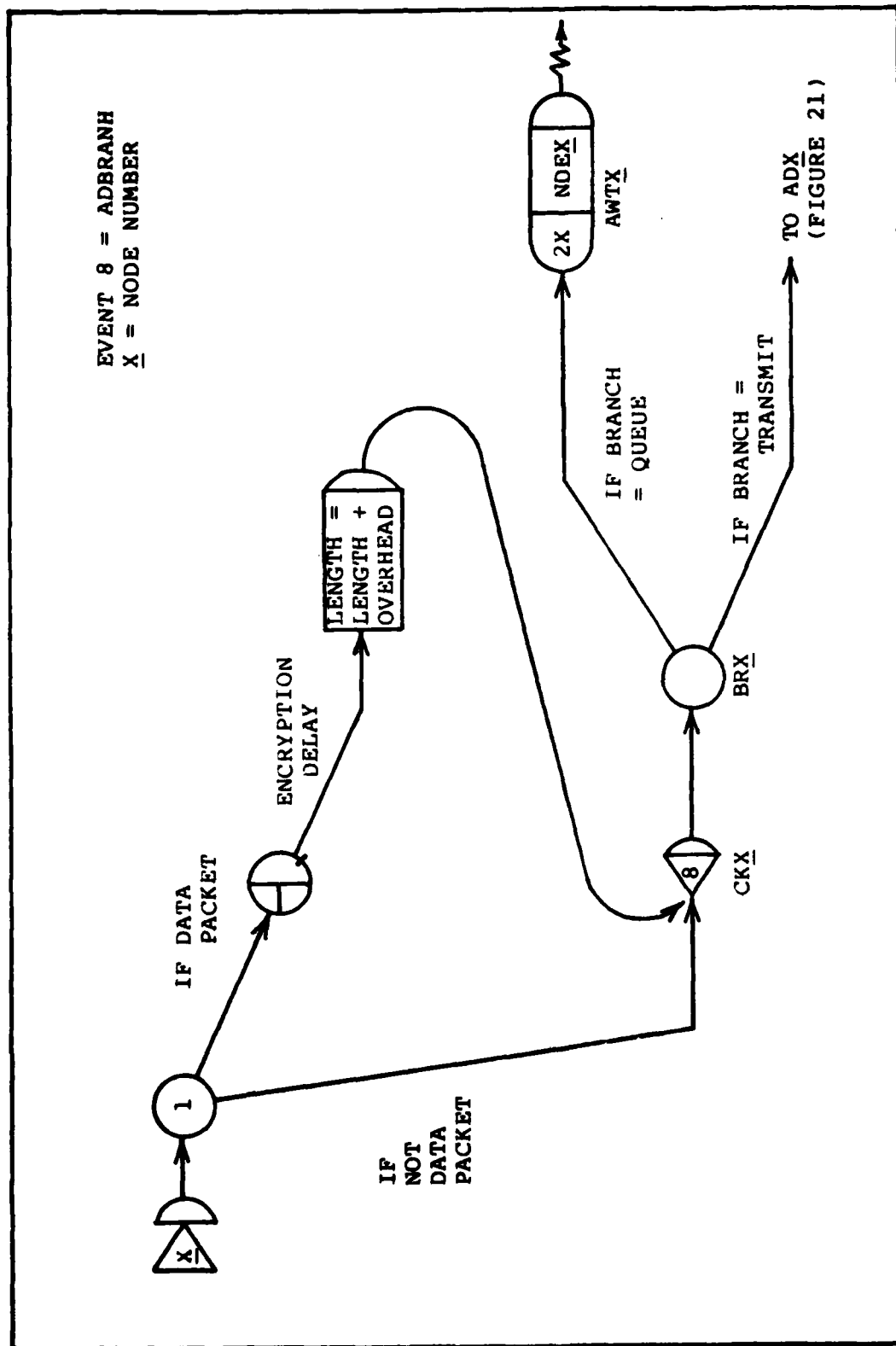


Figure 19. Processing For Entry to Node.

Data packets entered from MAKPKTS are branched from a GOON node to a QUEUE node for a processing delay such as encryption. The packets flow over an encryption delay activity to an ASSIGN node to have their packet length updated with packet overhead. They then flow to Event 8 where a decision on node entry is made and stored in packet attribute (13).

Acknowledgement packets from subroutines GNEGNAK and GENACKS flow directly to Event 8 from the first GOON node. Acknowledgements thus bypass queueing and delay for processing. The acknowledgements are routed through this structure to show that they must be constrained by the same resources as data packets and to save SLAM filing resources.

Regardless of packet type, the packets flow to GOON node BRX to test ATRIB(13) for branching according to the decision that was made in EVENT 8. If the packet may not enter the network node, it flows to AWAIT node AWTX which is a closed gate. It is filed until future discrete events remove it. Alternately, packets that may enter the network node are branched over an activity that leads to Node Acceptance of Packets (Figure 21).

4. Branching of Packets into Node (Figure 20). Subroutine CKADMT enters packets into ENTER node 7. CKADMT is called when buffer resources are changed in a subroutine. CKADMT evaluates packets waiting in the AWAIT node of Figure 19 for node admission. If entry is allowed, the packet is removed from the AWAIT node file and then entered in ENTER node 7. The

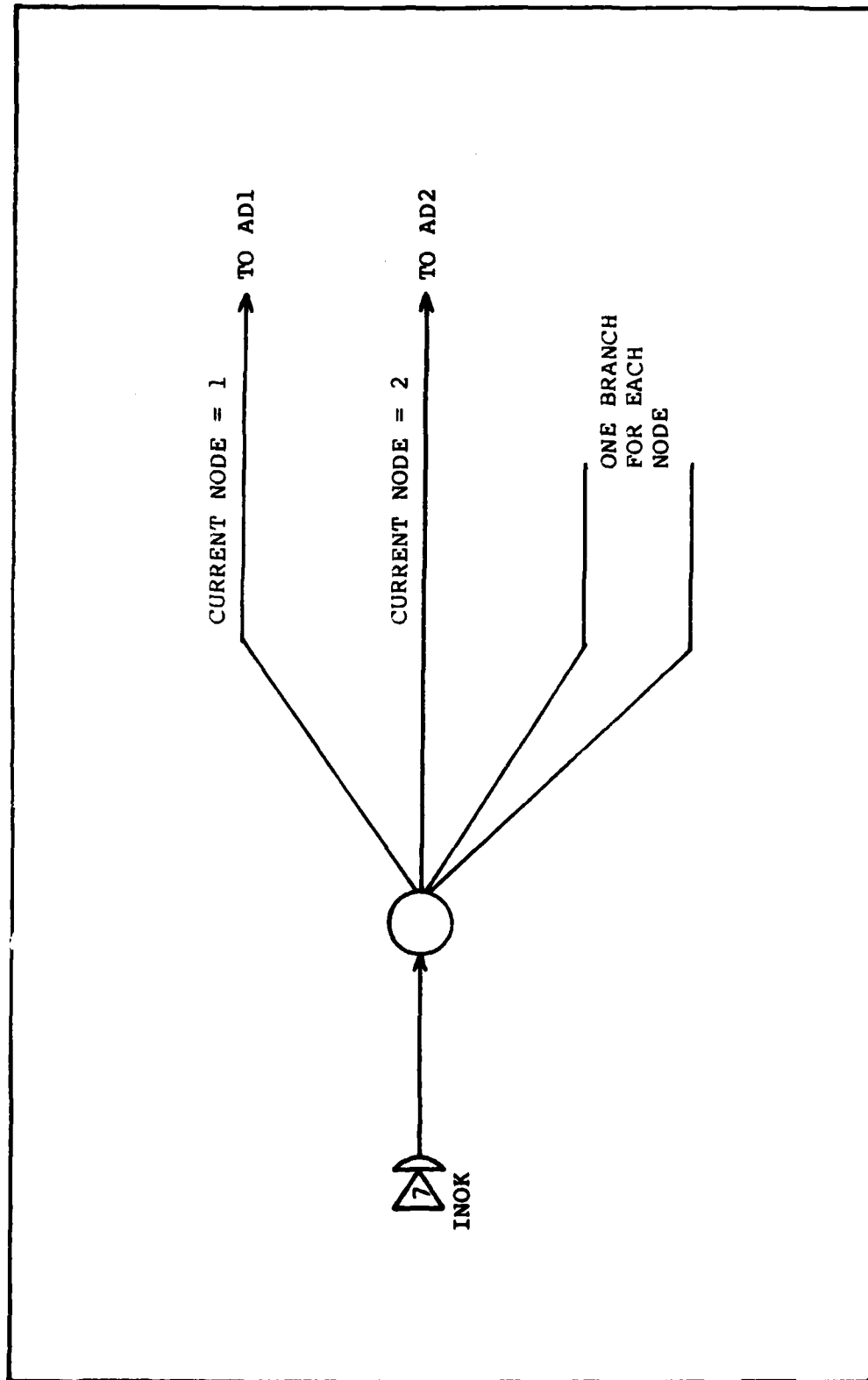


Figure 20. Branching of Packets Into Node.

packet then flows to a GOON node which branches it to the host node transfer process of Figure 21.

5. Node Acceptance of Packets (Figure 21). Packets enter this structure from Figures (19 and 20) to start the host-node transfer. GOON node ADX is used to route acknowledgements around the transfer activity. The time of node entry of the packet is assigned, and the packet flows to EVENT node 1, subroutine QUETOTX.

6. Server Queueing and Starting (Figure 22). Packets enter this structure from subroutine QUETOTX. The packet flows to Event 24, subroutine FILEPKT, which enters the packet in the appropriate server file. An entity returns from EVENT 24 to flow to an ASSIGN node for all models but ring models. The ASSIGN node is used to assign SLAM integer variable II with the number of packets in the server queue that the packet is filed. (II is used to provide an integer argument to SLAM function NNQ.)

If the network is a bus model and the packet just filed by EVENT 24 is the only packet on file, EVENT 15, subroutine BUSTART is called. For mesh models, EVENT 11, subroutine STARTTX, is called. BUSTART and STARTTX are both mechanisms for determining if transmission should be started. They also get a packet from the appropriate file and start it if starting conditions are met. (Ring models depend on timing loops that call controller subroutines to perform these functions.)

7. Reentry Into Network For Packet Transmission (Figure 23). All packets that enter transmission from event subrou-

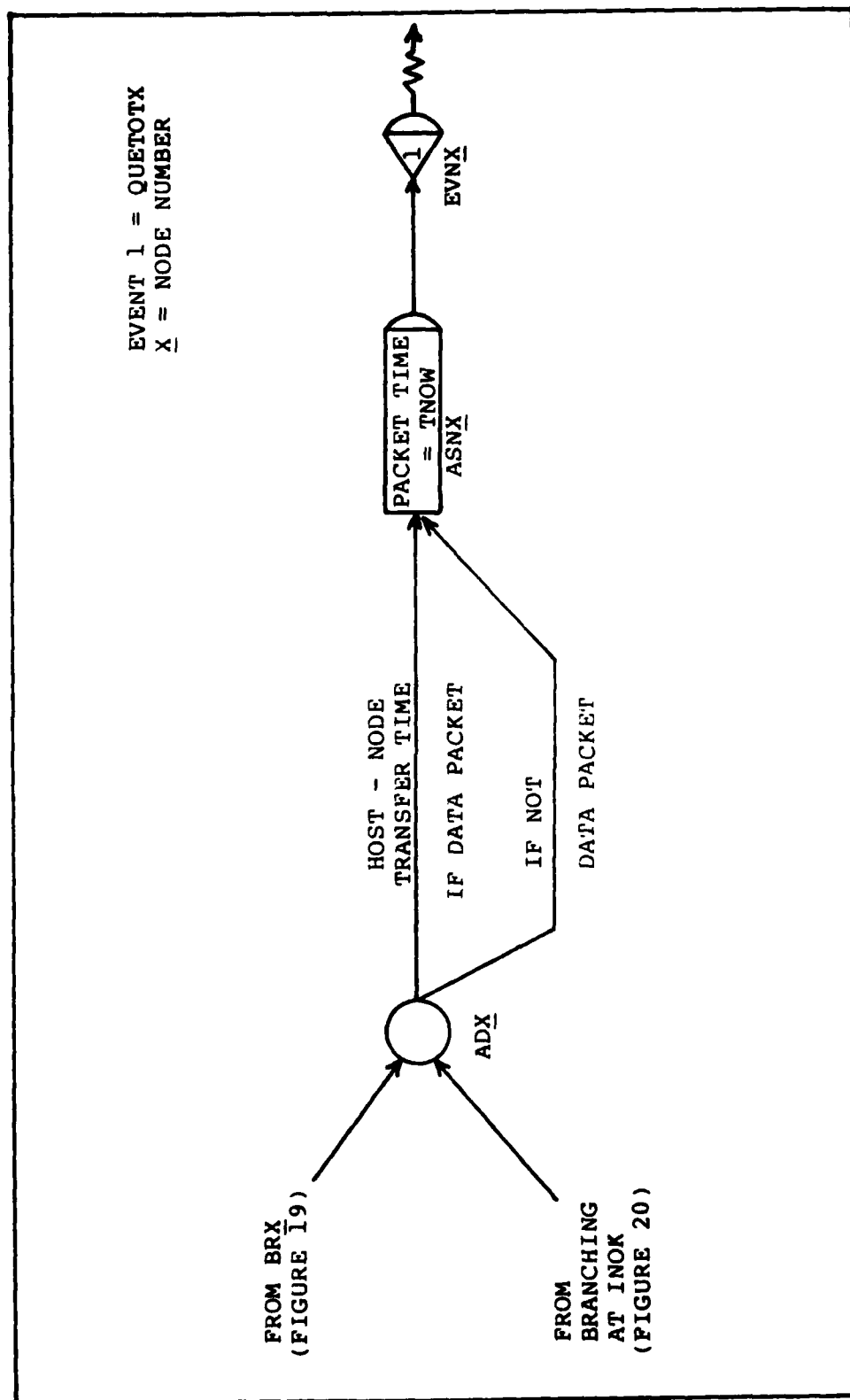


Figure 21. Node Acceptance of Packets.

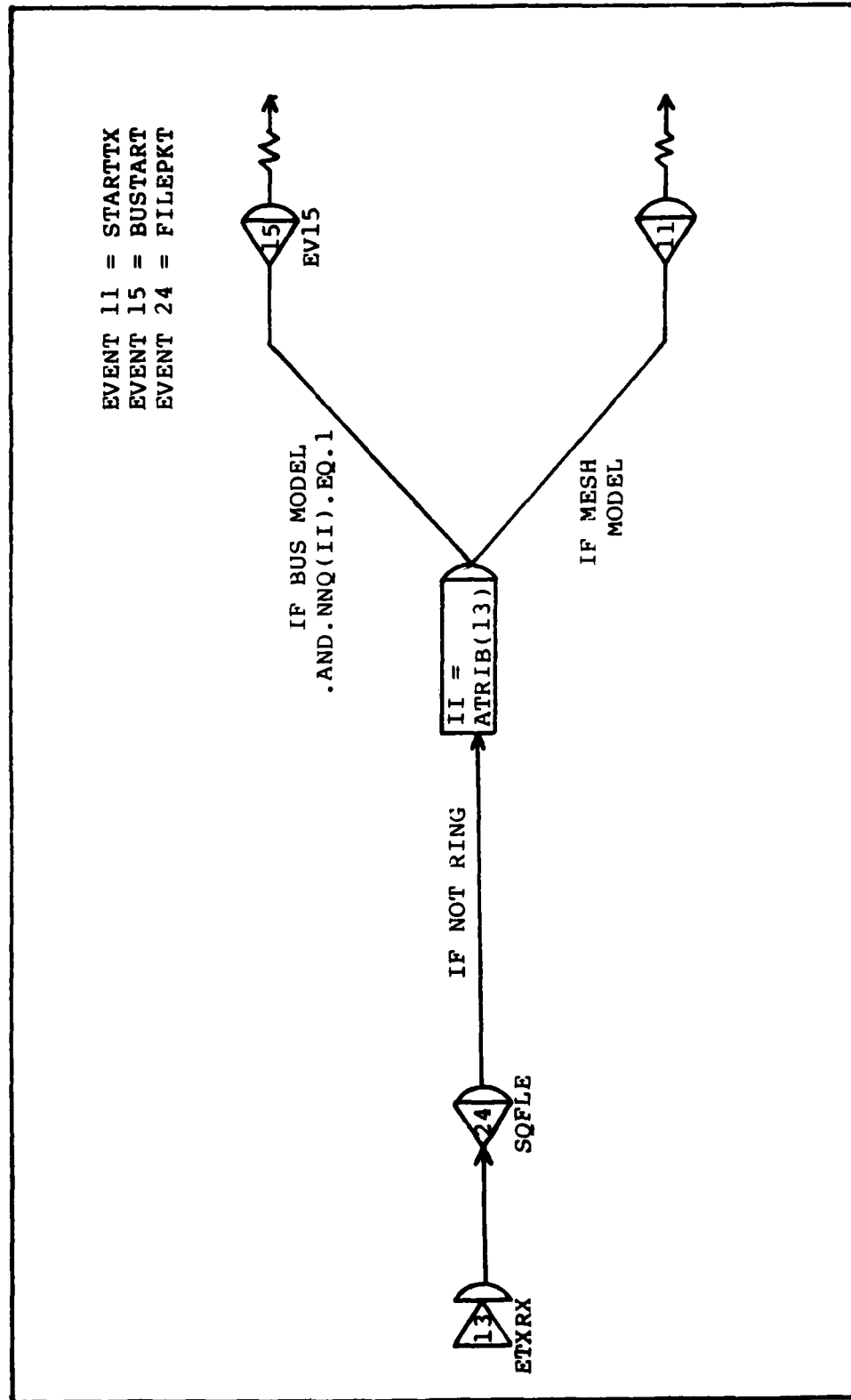


Figure 22. Server Queueing and Starting.

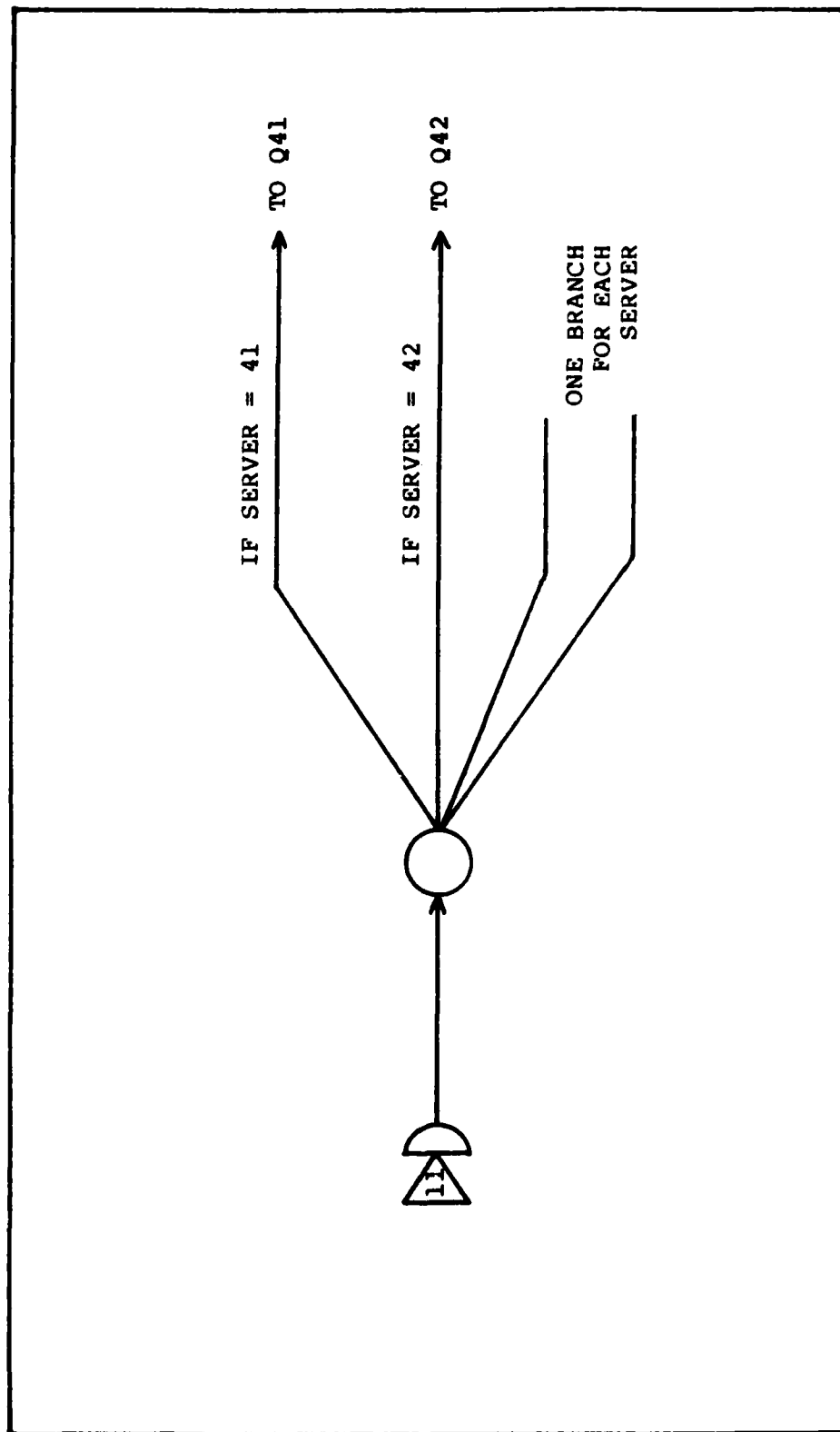


Figure 23. Reentry Into Network For Packet Transmission.

tines do so through ENTER node 11. The packets flow to a GOON node to branch to the proper transmission structure. The entities flow over activities to GOON nodes QZ that are the entry point into the Transmission structures (Figure 24).

8. Transmission Structure (Figure 24). This SLAM network structure models the transmission process with a great deal of flexibility. It is admittedly somewhat complex, but models a great deal. The first activity branching from GOON node QZ is simply a timer to EVENT node 5 (subroutine STOPACT). The ASSIGN node marks the entity for possible removal from the SLAM calendar at a future time. An entity flows on this branch to represent the full transmission time of a packet. STOPACT is an event which terminates the activity on the third branch from GOON node QZ. The third branch represents the transmission activity which is keyed on STOPA(II). (The ASSIGN node is necessary to produce argument II.) It is stopped by STOPACT either when the first branch completes its timing or whenever any other process calls STOPACT for that server from within the discrete event subroutines. Thus, transmission can be stopped at any time, even if the stopping time is not known when transmission is begun.

The second activity branching from GOON node QZ is taken if a transmission error exists and the immediate error detection scheme is specified. The entity on this activity takes a time equal to the simulated time to error detection at the destination node. The entity on this branch is "marked" so that subroutine RECVPKT will be able to make the decisions required

to respond appropriately to it. RECVPKT will ignore the packet that flows from the third branch which still takes the normal transmit time. (This feature is included for mesh networks using immediate detection of errors.)

The events that occur following the activity on the third branch perform several functions. Depending on the network type and packet type, the following decisions and actions occur in EVENT 9, subroutine FREERSC. For ring models, origin node buffer resources are freed. For other models, the server is cleared. After EVENT 9, EVENT 10 or 11 may be executed depending on network type. Both of these events (subroutines STARTTX and RESTART) are concerned with restarting the server that just ended transmission of the last packet.

All packets flowing on the second and third branches from GOON node QZ eventually reach GOON node CKRX. From CKRX, the packets are allowed to flow on the next activity only if the destination node buffer has room for the packet, otherwise, it is lost to the system. When packets are allowed to flow, the activity represents the propagation delay to the next node.

After that propagation delay, the packet reaches EVENT 2, subroutine RECVPKT, for processing at the destination node.

The last activity branching from GOON node QZ is taken under complex conditions evaluated in USERF(10). After a packet is released to flow on this activity to GOON node FMNX, it is branched two ways. Over one activity, it enters an AWAIT node which is a closed gate RTXX. The closed gate is used to file the packets in file 3X to model the retransmit queue. The

second branch from FMNX essentially starts an activity that represents a time out for the packet just entering the retransmit queue. The ASSIGN node marks the entity for possible removal from the SLAM calendar. After the retransmission activity times out, EVENT 3, subroutine RETXMT, will begin the retransmission process.

9. Entry of Packets into the Retransmission Process (Figure 25). Packets enter this structure from subroutine ENTRRTX. ENTRRTX removes the packet from the server queue after transmission is complete and starts the retransmission process at enter node 12. Packets then flow to a GOON node for branching to the respective node retransmission process. (This structure is not used in ring models.)

10. Hold Queue Selection, Entry, and Time-Out (Figure 26). This structure is very similar to the retransmission process of Figure 24 and the branching structure of Figure 25. When packets are to be filed in the hold queue, subroutine HOLDPKT enters it into ENTER node 10. The packet then flows to a GOON node for branching to an AWAIT node which represents the appropriate hold queue. An entity also flows from the GOON node over an activity representing the hold-time-out. The ASSIGN node marks the entity for possible removal from the SLAM calendar. When time-out occurs, EVENT 6, subroutine HLDTOU, retrieves the packet and enters it back into transmission.

11. Message Reassembly Files (Figure 27). These AWAIT nodes are declared in the network but no routing is done to

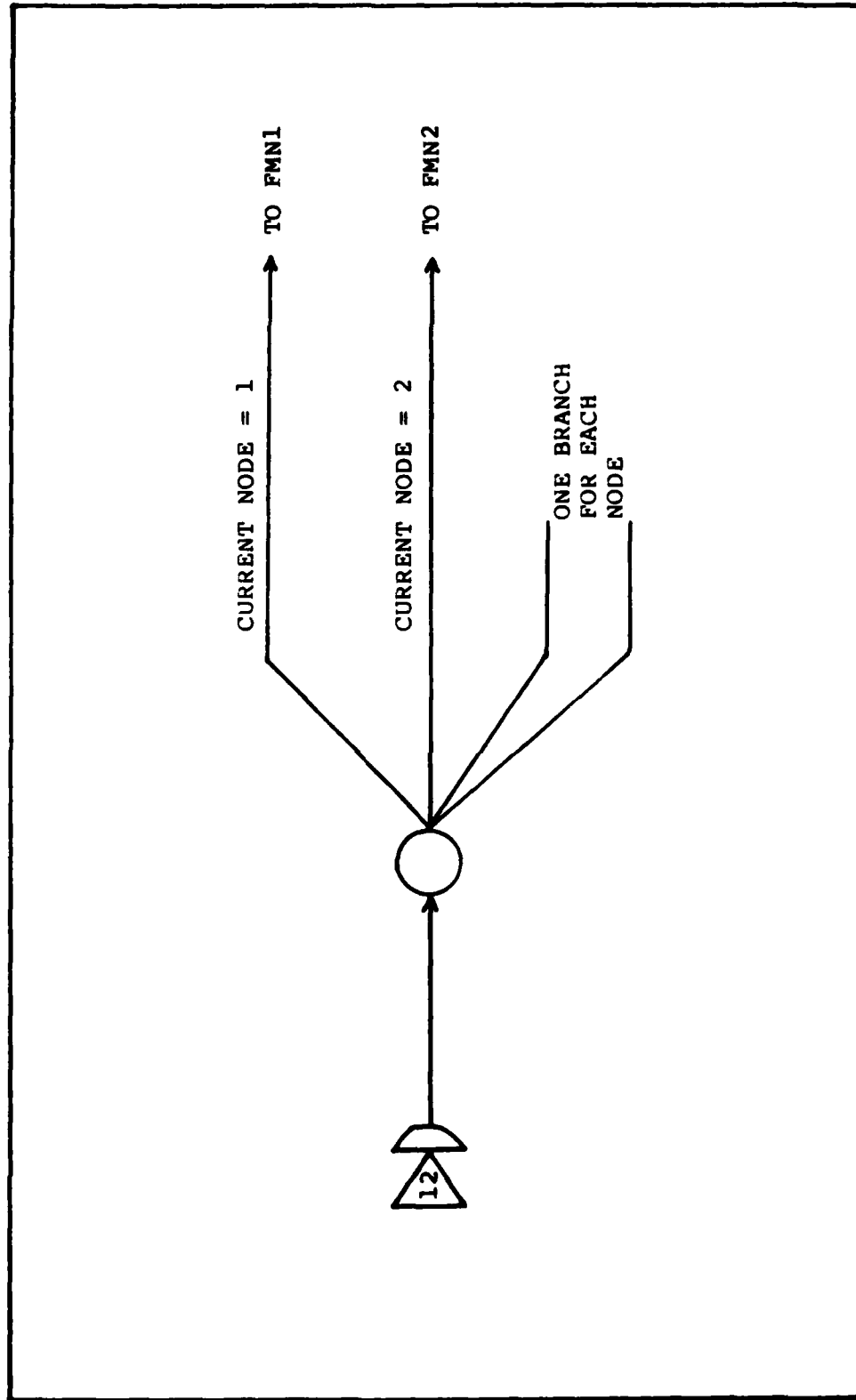


Figure 25. Entry of Packets into the Retransmission Process.

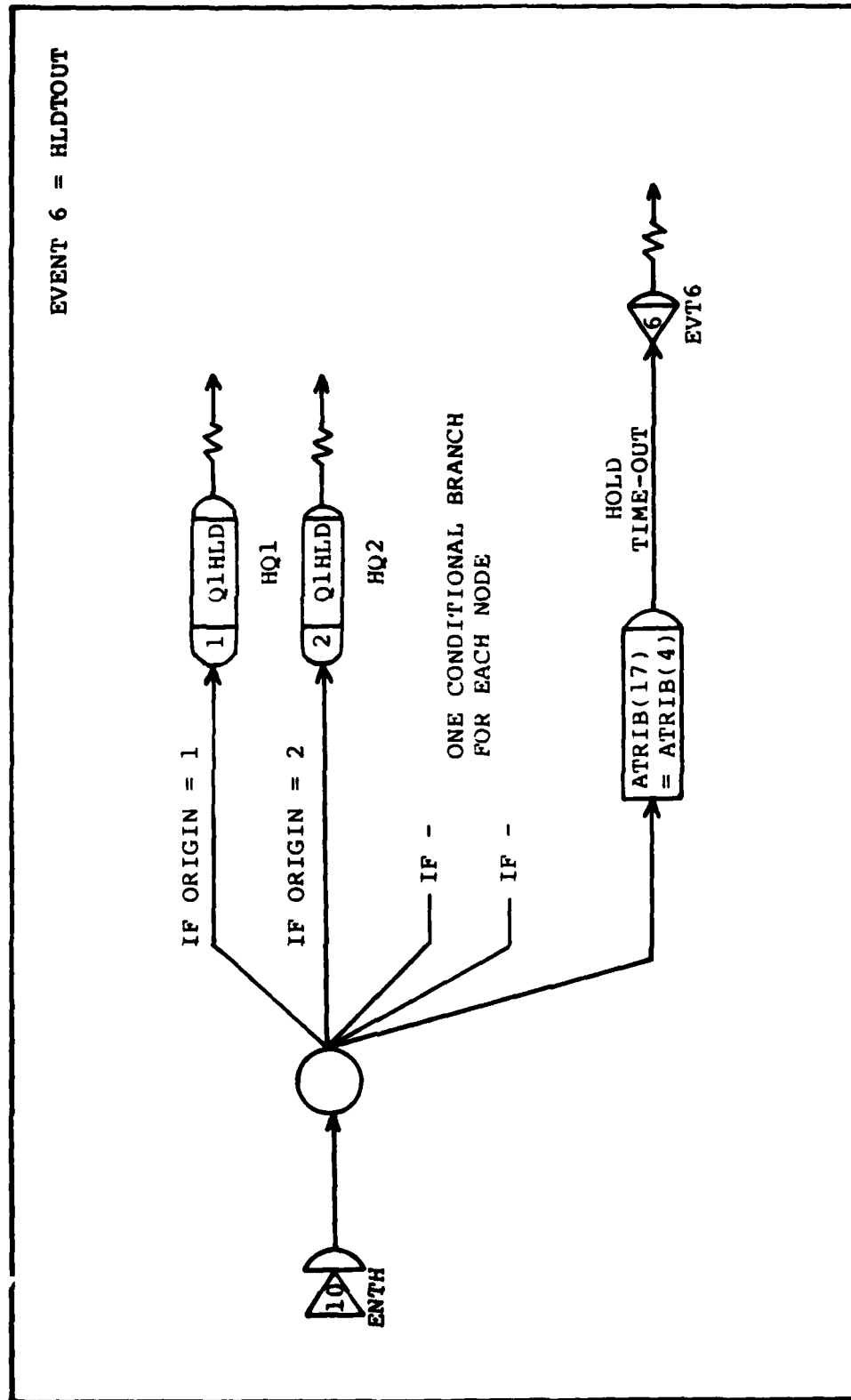


Figure 26. Hold Queue Selections, Entry, and Time-Out



RSM1



RSM2



RSM3

.

ONE GATE
FOR EACH
NODE

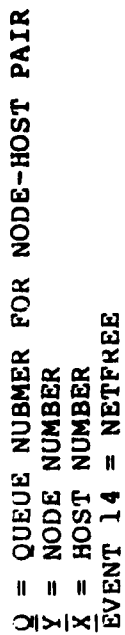
Figure 27. Message Reassembly Files.

them in the SLAM network. The files associated with these AWAIT nodes are all manipulated by subroutine ENFILE.

12. Flow of Completed Messages from Node to Host (Figure 28). Completed messages from subroutine ENFILE are entered at ENTER node 14 to begin their transfer and processing at the host. Packets are assigned a branching index via USERF(2). (USERF(2) is currently based on a homogeneous assignment of two hosts at each node.) The packets then flow to a GOON node that uses that index to determine the branching to the appropriate QUEUE node for each node-host pair. The QUEUE node files packets prior to their flow over the node-host delay and transfer activities. Following their transfer, EVENT 14, subroutine FREENET, is called to free the node buffer by the length of the message. The message then flows over an activity representing decryption and arrives at a COLCT node where message arrival statistics are collected. The message's data compression is reversed by an ASSIGN node just prior to termination from the system.

13. Initiation, Selection, and Cycling of Ring Controllers (Figure 29). An entity is created at time zero and flows to a GOON node if the ring network is selected. The entity is then branched to one of the controller cyclers based on the ring controller specifications.

The token ring model is based on EVENT 10, subroutine CONTRL 1. However, before EVENT 10 is entered, attribute 13 must be initialized to zero in an ASSIGN node. Subroutine CONTRL 1 controls all decisions and actions concerning



110

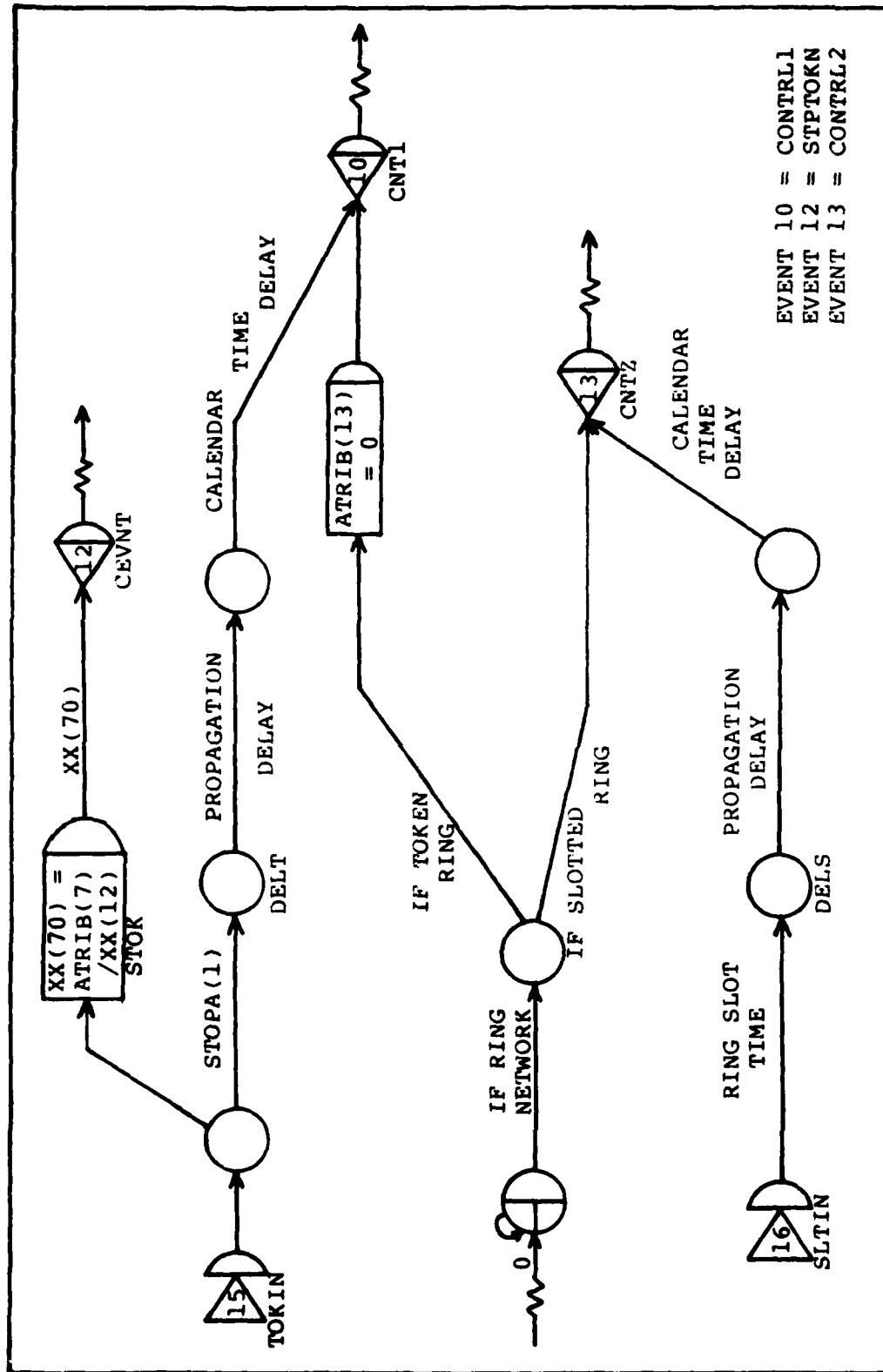


Figure 29. Initiation, Selection, and Cycling of Ring Model Controllers

node-to-node transmission and enters an entity representing the TOKEN at ENTER node 15. The TOKEN flows to a GOON node where it is committed to an activity keyed on STOPA(1). That in turn is controlled by the other activity that branches from the GOON node. The branch on top essentially uses an ASSIGN node to assign the TOKEN transmission time to a SLAM XX variable. That variable is then used as the next activity time as the entity flows to EVENT 12, subroutine STPTOKN. STPTOKN is used to terminate the activity keyed by STOPA(1). The TOKEN is thus released from that activity and arrives at GOON node DELT. It flows over an activity representing propagation delay and a final GOON node and activity. This final activity before EVENT 10 is not part of any modeling concept but an accommodation to the SLAM system. This TOKEN ring cyler is synchronized with the transmission processes in the network. The packet reception processes must be completed prior to EVENT 10. The calendar time delay ensures that EVENT 10 is executed a minute time after the reception processes are finished.

The slotted ring model is based on EVENT 13, subroutine CONTRL2. If the slotted ring model is selected, an entity flows to EVENT 13 to begin cycling of the slotted ring controller. Subroutine CONTRL2 controls the decisions and actions concerning node to node transmission. it also enters an entity in ENTER node 16. The entity then flows over activities that represent the ring slot time, propagation delay, and a calendar delay time. (The calendar delay time is used for the same reason that it is with the token ring model.) After flowing

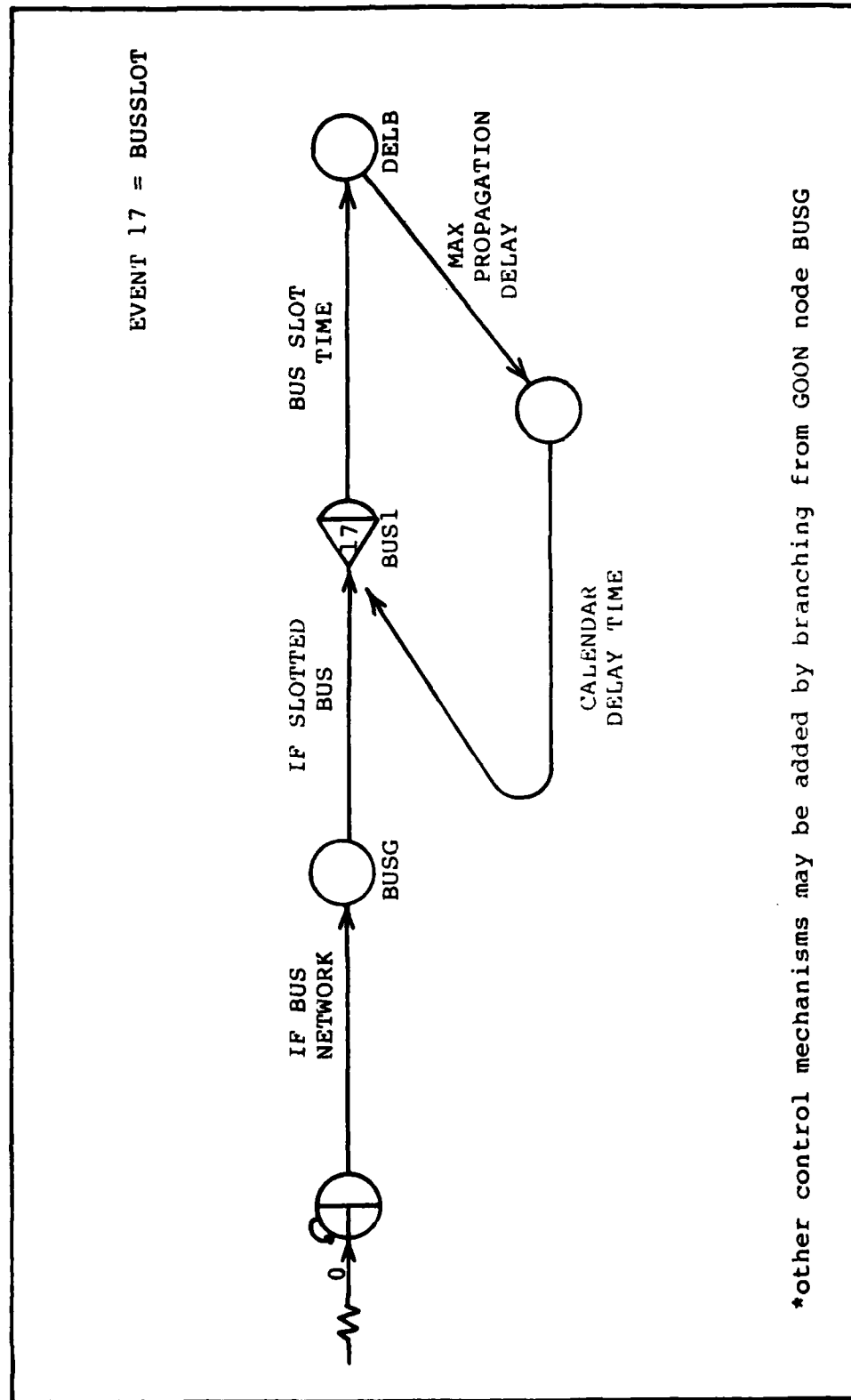
over these activities, subroutine CONTRL2 is executed again when the entity arrives at EVENT 13.

14. Bus Arbitration Controller(s) (Figure 30). This structure is very similiar to the ring network controllers. An entity is created at time zero and flows to GOON node BUSG if the bus network is specified. The BUSG node is included as a branching point for other arbitrator cyclers that could be added by a user. For the slotted bus model, an entity flows from BUSG to EVENT 17, subroutine BUSSLOT. After BUSSLOT handles transmission processing, the entity flows over activities that represent the SLOT transmission time, a maximum propagation delay specified by the user, and the calendar delay time. After flowing over these activities the entity arrives at EVENT 17 to repeat the cycle.

15. Statistics Collection Cycler (Figure 31). This structure is initiated just as the previous two. At time zero an entity is created that flows over an activity representing the first interval of statistics collection. The entity enters EVENT 25, subroutine PRTSTAT, to print selected statistics for selected analysis. Next, the entity enters EVENT 21, subroutine STATCOL, to collect statistics on network performance over the collection interval. After STATCOL, the entity flows over an activity representing the statistics interval and enters both EVENTS again.

Summary

This chapter gave an introduction to the SLAM network orientation. It also described how EVENT and ENTER nodes are



*other control mechanisms may be added by branching from GOON node BUSG

Figure 30. Bus Arbitration Controller(s)*

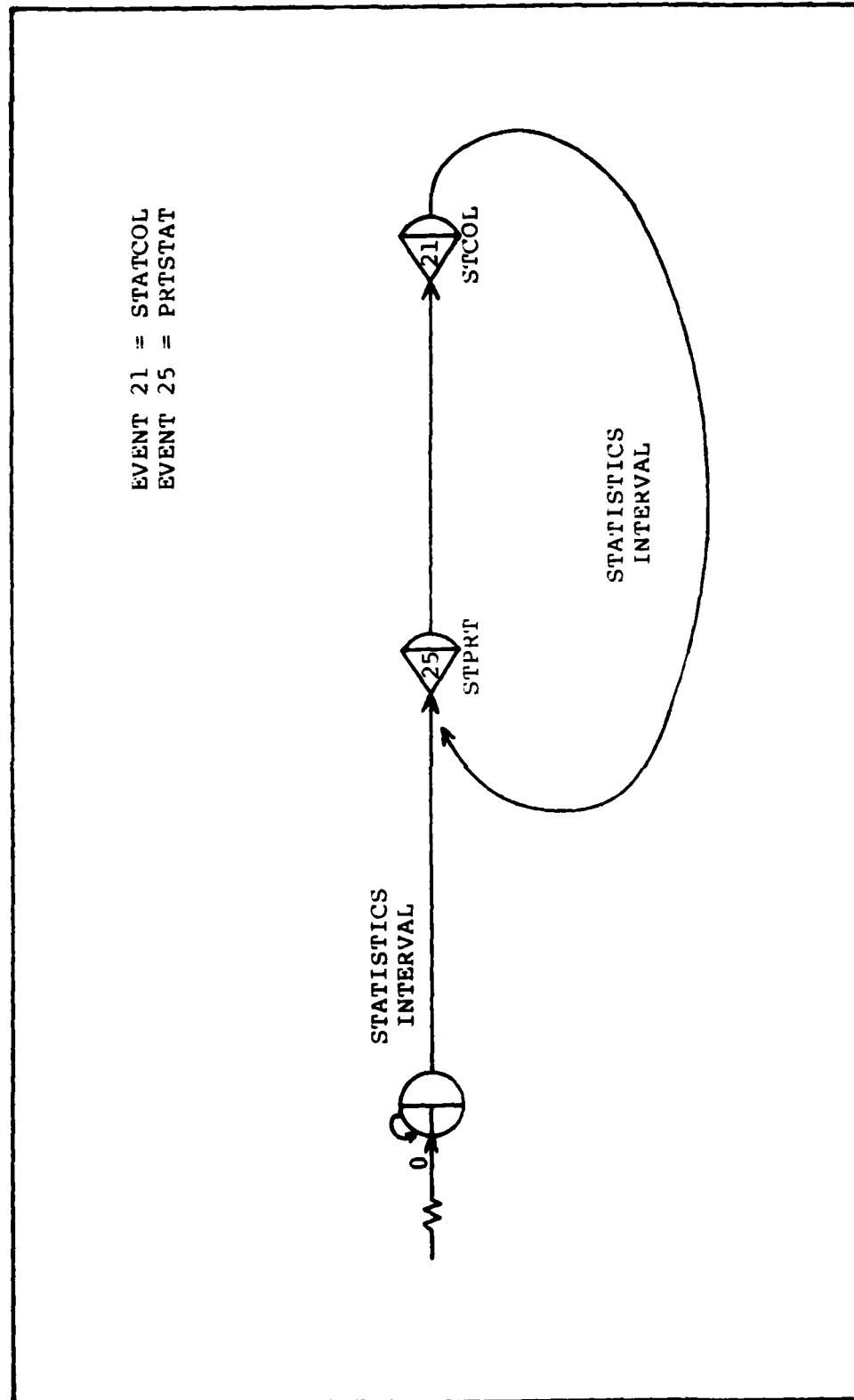


Figure 31. Statistics Cycler.

used to connect the network and discrete events of the simulation system. Each structure that was implemented in the SLAM network was then presented in detail.

The sequence of development of the simulation system, including problems in implementing network structures and discrete events, is given in the next chapter.

Appendix to Chapter V

Functions of EVENT Subroutines

This appendix is an abbreviated reference of EVENT subroutines. Full documentation of all subroutines is given in Appendix A.

A concise statement is given for each subroutine indexed to a SLAM EVENT. They are ordered according to their appearance in Figures 17 through 31.

Figure 17

EVENT 4 Subroutine MAKPKTS. MAKPKTS checks network status to suppress a message if necessary and performs two primary functions. It makes several assignments to message attributes and enters the message into the SLAM network packet-by-packet.

Figure 19

EVENT 8 Subroutine ADBRANH. ADBRANH checks the node buffer and other flow control criteria for packet admission into the node. The decision is assigned to attribute 13 to control branching when the entity returns to the network.

Figure 21

EVENT 1 Subroutine QUETOTX. QUETOTX assigns next node, server index, and error attributes to each packet before entering it into the SLAM network. Based on an attribute flag, QUETOTX will enter the packet into the network at ENTER node 13 for queueing for transmission, or ENTER node 11 for immediate transmission.

Figure 22

EVENT 11 Subroutine STARTTX. STARTTX is used only by mesh models. If a server queue has a packet waiting, and the server is idle, STARTTX enters the packet into the network for immediate transmission.

EVENT 15 Subroutine BUSTART. BUSTART is also responsible for starting the transmission process but for bus models only. Depending on the arbitration status of the node, BUSTART will start the packet or schedule the packet for transmission.

EVENT 24 Subroutine FILPKT. FILPKT simply files the packet in the appropriate server queue.

Figure 24

EVENT 2 Subroutine RECVPKT. RECVPKT is a general routine that filters out errors, collisions, and packets types to perform all processing at the destination node.

EVENT 3 Subroutine RETXMT. RETXMT is called after a retransmission time-out to retrieve a packet from the retransmit queue and begin the transmission process again.

EVENT 5 Subroutine STOPACT. STOPACT stops the packet in transmission on the activity keyed to the appropriate server index.

EVENT 9 Subroutine FREERSC. FREERSC handles the node buffer deallocation for ring network models. For the mesh and bus models it handles the removal of the packets from the server queue after transmission is complete.

EVENT 11 Subroutine STARTTX. See statement for Figure 22.

EVENT 16 Subroutine RESTART. RESTART schedules transmission of the last packet again if the prior attempt resulted in a collision at the transmitting node. Otherwise, actions are taken to start the transmission of the next packet.

Figure 26

EVENT 6 Subroutine HLDOUT. After a hold-time out, HLDOUT removes the packet from the hold queue and reenters it for transmission queueing.

Figure 28

EVENT 14 Subroutine NETFREE. NETFREE frees the network node buffer resource by the length of the message that was just transferred to a host.

Figure 29

EVENT 10 Subroutine CONTRL1. CONTRL1 handles the decision making for starting transmission of packets according to the token ring protocols. It enters packets into the network for transmission. It also enters an entity representing the token into the token ring controller.

EVENT 12 Subroutine 12 STPTOKN. STPTOKN causes the activity upon which the entity representing the token is flowing to stop. (The token then flows over other activities.)

EVENT 13 Subroutine CONTRL2. CONTRL2 handles the decision making for starting transmission of packets according to the slotted ring protocol. When appropriate, it enters packets

into transmission and another entity into the slotted ring
cycler to continue the cycle.

Figure 30

EVENT 17 Subroutine BUSSLOT. BUSSLOT handles decision
making for starting transmission of packets according to the
slotted bus protocol. At present, it starts transmission at any
node that has a packet on server queue.

Figure 31

EVENT 25 Subroutine PRTSTAT. PRTSTAT prints the
throughput (number of packets) and average packet delay time,
over the previous statistics collection interval.

EVENT 21 Subroutine STATCOL. STATCOL collects
statistics on throughput, transmission errors, and collisions
over the previous statistics collection interval.

VI. Implementation

This generic program was implemented and tested on the ASD Cyber computer system using Intercom. This chapter covers the sequence of development of the program, problems encountered during this phase of the project, and a description of the specific models chosen for demonstration purposes.

Sequence of Development

The first versions of the program were written mostly in the SLAM Network orientation. This permitted a simple conceptualization of the overall problem to begin with. At first, few structures were modeled and parameters were greatly simplified. For example, only three nodes were included in the initial model, each having one host. Additionally, SLAM RESOURCES alone were used to control processes that would later be developed into flow control protocols.

These first networks were verified using SLAM Traces augmented with user-provided instrumentation before further development. Once the framework for the program was developed, primarily using the SLAM network orientation, further development consisted of the addition of discrete events. In some cases, the addition of discrete events could be viewed as adding building blocks to the model. In other cases they replaced SLAM network structures which were too limiting to the model. Some examples of building block discrete events are the controllers for the ring network models, subroutines CONTRL1

and CONTRL2. These were added after the message and packet flow through the network were established. They added the specific modeling required for the token and slotted ring networks respectively.

An example of SLAM network structure replacement with discrete events is the implementation of flow-control. The SLAM RESOURCES were used initially to model the network node buffer resources. These resources were the only criteria for the admission of packets into the network nodes. Since advanced models require other parameters or constraints to control admission to the node, the SLAM RESOURCES were inadequate. So, as entities move through the network, they encounter EVENTS such as FREERSC, NETFREE, and ADBRANH which were added to make decisions regarding flow through the network as well as control the (de)allocation of node buffer resources.

A major milestone was the implementation of greatly simplified models of mesh, ring, and bus networks. To get these simplified models up, basic link protocols were necessary. These link protocols are based on different packet types and the subroutines that generate or handle them. The development of node-to-node, end-to-end, and ring acknowledgements was the basis of this development stage.

The next major thrust was the introduction of greater levels of throughput and random processes into the models. This served to test the flow control elements of the model. Additionally, the dynamics of the randomized, high-level traffic tested many parts of the model as well as the SLAM system.

During this stage of development, errors such as the misfiling of packets were revealed through reference to the SLAM Summary Reports. High throughput also revealed a necessary overhaul of the retransmission process that will be mentioned in the problems section.

Problems

There were several problems that were dealt with during the course of implementation and testing. They can be classified as problems associated with the SLAM environment and problems in software design.

SLAM Environment. This is not a general discussion of limitations of SLAM. This section covers difficulties in implementing a generic program in SLAM and problems that may have been avoided by more specific SLAM documentation.

With a generic program, arbitrary assignment of numbers to files could make it easier to implement and expand a program. However, most versions of SLAM have a maximum of 100 files that can be used. Additionally, the file numbers are sequential from one to the highest file number requested by the user. All files in that range are available regardless of the need or use in the program. The program developed in this project determines file assignment with simple arithmetic operations. The maximum of nine network nodes available in this simulation program is a result of this characteristic of SLAM and the file assignments used in this program.

The SLAM system also imposes upper limits on the number of

ENTER nodes and the number of GATES that may be used. The maximum of each of these structures for the SLAM version used in this project (SLAM II, Version 1, Release 3) is 25. These limitations are not included in Pritskers documentation (11) because they are version specific. Unfortunately, the existence of such limits is not mentioned either. As a result, both of these limitations forced software redesigns in the project.

Another constraint imposed by the SLAM system is the index (NTC) of the STOPA subroutine. The index used for NTC in the version used for this project may not exceed 50. That constraint, or its existence, is not mentioned by Pritsker (11).

SLAM network TERM nodes also have a poorly documented characteristic. The "TC" for TERM node is a specification by the user for a maximum number of entities to flow through that term node. Arrival of the TC'th entity causes the simulation run to be terminated. The documentation (11) indicates that TC is defaulted to infinity. Consultation with Pritsker and Associates revealed that the default is not infinity but is dependent on the version of SLAM being used. This can be a problem if long runs with high throughput are expected. Fortunately, specifying a very large TC on TERM nodes will override the default.

User functions used to return activity times to the SLAM network must also be handled carefully. If a user function returns a negative value to the network for an activity time, the activity time will be zero. So, if a user function faulty, the SLAM system will sometimes disguise the problem by

continuing execution rather gracefully rather than failing.

The SLAM calendar can impose a great deal of overhead in terms of file space. An early implementation of the retransmission process caused an inordinate number of RETXMT EVENTS to build up on the calendar. When an acknowledgement was received, the packet in the retransmission file was removed, but the activity representing time-out remained on the calendar. RETXMT, (executed after time-out), was designed to do nothing if the packet was already removed, so there was no fatal problem. However, in conditions of high throughput, the program invariably ran out of file space. For time-outs such as that preceeding RETXMT, a routine was developed to remove the activity representing the time-out from the calendar when the appropriate acknowledgement was received.

Software Design Considerations. The following comments relate to considerations which are not problems in themselves, but must be dealt with to prevent problems. They also show how using SLAM, especially in the combined Network-Discrete Event orientation, affects the design process by requiring additional decisions.

The first feature of the SLAM system that must be grasped when using the combined Network-Discrete-Event orientation is the ATRIB array. It is simply an array that may be referenced from either the network or from discrete events. It is the only array that may be referenced from the network and is stored in the SLAM calendar with the current ATRIB values when an entity departs any network node. The discrete events may use any

number of data structures including the ATRIB array. The integrity of the ATRIB array is an important consideration in any discrete event and especially in subroutines called by discrete events. It is very easy to corrupt the ATRIB array before returning to the SLAM network. Maintaining the integrity of the ATRIB array can be accomplished by using additional arrays in subroutines. These arrays can be used to protect or save the ATRIB array. They can also be used and as an alternate way to return entities to the SLAM network. An entity can be created in an array from part of the ATRIB array or none of it at all. The created entity can then be entered in the network or scheduled for another event easily by the SLAM ENTER and SCHED subroutines. These routines accept any array as well as ATRIB. An example of an entity being created using another array is the "TOKEN" array. The "TOKEN" created in subroutine CONTRL1 for the token ring controller loop aided in maintaining the integrity of the entities that represented packets.

The network-discrete event orientation gives the modeller flexibility in implementation. However, the SLAM programmer must often decide if a particular part of a model should be implemented in a SLAM network structure, or in discrete event subroutines. The choice can be difficult if both orientations can be used. The SLAM network offers the advantages of clear, graphic conceptualizations. It also can be very powerful. However, as the modeling requirements become more complex or advanced, the limitations of the network structures force implementation in discrete event subroutines. This can be a

problem if the transition requires major redesign.

Flexibility in this combined orientation requires careful consideration of where to implement decisions. Decisions can be made on network activities or in subroutines. The problem is that decisions made in one orientation are not apparent to the programmer when he/she is looking at the other orientation. For example, when developing or tracing through a subroutine in this project, the decisions made in the network must be well-known. In several cases, conditions on activities preceeding EVENT nodes were used to prevent entities from flowing to the EVENT nodes. In such cases, the subroutines don't need to check for special conditions. On the other hand, if checking a condition is not performed in either orientation, the results can be very confusing.

The three main topology types used in the design of this project require many decisions in both the network and discrete events. The flow of entities through the network depends on many criteria as well as the type of model that was chosen. As a result, tracing the flow can become confusing. This is accentuated when combined with different responses of discrete events that also make decisions based on many criteria as well as the type of model. So, when tracing the flow of an entity through the network, the type of model chosen must be continually remembered to sort through the different paths the entity may take.

Models Chosen for Demonstration

The models chosen for demonstration and some general comments are given below. The specific development of the models, analysis, and discussion are given in Appendix B.

MESH. To demonstrate a mesh network, some of the parameters used in computer communication networks such as the ARPA network were selected (15). A comparative analysis with the ARPA network was not feasible because of the limited number of nodes possible with this simulation program. Additionally, a valid analysis would have required significant additions to the program. As a result, the mesh model as demonstrated provides results which can be used only to analyze changes to the model itself. A relative comparison of the different parameters or options can then be made.

RING. The token and slotted (one slot) rings from Bux (4) were modeled with this program. They were implemented with subroutines CONTRL1 and CONTRL2 for the token and slotted models respectively. The models developed closely parallel Bux's except that 5 nodes were used instead of 50. Also, the node latency had to be carefully adjusted for the slotted model.

The runs of these models and those to follow were stepped through intervals of increasing offered traffic. A data point of throughput versus delay is generated for each interval. This data is normalized for comparison with the Bux paper.

BUS. A CSMA/CD bus similar to that in the Bux paper was modeled. The model once again used only 5 nodes versus 50

in Bux. Also, there were two other significant differences from the Bux model. The demonstration model uses an acknowledgement scheme consistent with the design of this project. In terms of bits of transmission, 24 bits of acknowledgement packet were added for each data packet (1024 bits). However, the arbitration process is executed for each acknowledgement. Thus a difference in the analysis was expected and found.

The Bux model did not allow queueing of packets at the 50 nodes. To approximate that, an attempt to limit the number queued at the 5 nodes of this model was made. That is described further in Appendix C.

Summary

This chapter showed the sequence of development of the simulation system as a progression in SLAM orientation. The network orientation was emphasized at first, but gave way to more of the discrete event orientation as sophistication increased. Several problems that were experienced during development were identified. Knowledge of these problems may aid future users in further development of this system or other systems.

Several models that were chosen for demonstrations were described. The detailed development of those models and analysis from their execution is contained in Appendix C. Chapter VII is a summary and evaluation of the model development and analysis from the demonstrations.

VII. Results

This chapter is a summary and evaluation of the data presented in Appendix C. Appendix C is a presentation of the data produced by network models that were used to demonstrate the simulation system.

The evaluation performed in this chapter covers three points that are important to any potential user. The ease of modeling, validation of models, and use of resources by the simulation system are the global criteria that will aid the user in analyzing this system for his/her application.

A more general summary of these points follows the comments on each model.

Mesh Model

The mesh model developed for the demonstration was not validated against any other data. It was run to observe its own behavior and make comparative analyses using different conditions of offered traffic and packet error rates in transmission.

Ease of Modeling. The simulation system supports modeling of mesh networks well to a point. It was not difficult to develop a basic model, however, the detailed implementation of lower level protocols can be very difficult. The greatest deficiency is in flow control mechanisms. Even though several

flow control mechanisms are offered by this system they cannot hope to match the multitude of flow control protocols known or in use.

The maximum number of nodes modelable, nine, can also be a short coming. Many models may be satisfied with nine, however, the large computer-communication networks such as the ARPA network cannot be simulated effectively with this limitation.

Validation of Model. Given the limitations above, validation of mesh network models is difficult. The development of complex protocols such as buffer reservation systems, for example, would be a significant burden for the user of this simulation system. Development of details such as that would be necessary before attempting validation.

Use of Resources. The mesh model is a moderate consumer of computer resources. This is a subjective comment, and relative to other models demonstrated by this simulation system. The model that was demonstrated used a slower node to node transmission rate than desired (Ref Appendix C, mesh model development). This reduced the load on the simulation system in terms of numbers of packets, or entities, that would be required to yield high utilizations of servers.

A 500 msec interval of a simulation run typically required approximately 30 seconds of execution time. Lower transmission rates and single runs per job can be used to minimize the use of resources for this model.

Ring Models

Two ring models were demonstrated. They were taken from Bux (4) for comparative analysis and validation. The three point outline below covers the token and slotted ring models together.

Ease of Modeling. Both models were brought up with a minimum of development. Only one Fortran subroutine was required for each model. To drive the subroutines, which were EVENTS in the SLAM network, each model required a controller loop in the SLAM network (see Figure 30). There was no other model-specific development necessary.

The most difficult part of bringing up these models was the coupling of the subroutine with the SLAM network controller loop. Also, entities that flowed through the loop had to be carefully manipulated and isolated from the entities that represented packets.

Validation of Model. These models were implemented with five nodes versus fifty used by Bux. However, the manipulation of node latency produced equivalent models (see ring model development in Appendix C).

The data validated the token model with a high degree of confidence (note Figure 35 in Appendix C). The slotted model was validated but with less confidence (Figure 36). It required some interpretation of the data. Also, the data was difficult to accumulate at high throughput levels due to the great consumption of computer resources.

Use of Resources. Both ring models were definitely poor in this respect. Execution times for these models can exceed the simulation times by a factor of 100. Processing of the token or slot by each node of the ring represents several EVENTS which must be executed. Models such as these with a fast node-to-node transmission rate are executed very slowly.

Bus Model

The bus model demonstrated was a CSMA/CD bus also from Bux (4).

Ease of Modeling. The consideration of timing is much more complex in bus networks than in the others considered thus far. However, a base for modeling was developed in the ten subroutines for bus models in the simulation system. Many different buses can be modeled from these subroutines. The implementation of the CSMA/CD network access protocol for this model is based on subroutine CSMASRT. This is a complex scheme that did require significant development and testing. However, CSMASRT is a base for other CSMA/CD schemes of different persistence. Additionally, subroutines ARBITOR and ARBITOK are provided as the interface for other user developed non-CSMA/CD protocols. (These subroutines perform no function in their current configuration.)

Validation of Model. There were two significant differences that had to be considered in the validation of this model. As in the ring network, this model uses five nodes instead of fifty. Since the Bux model didn't queue packets, a

restriction on queueing in the five nodes served to approximate the fifty nodes. However, the most important difference between the models was the acknowledgement scheme used in the demonstration. A simplified argument is given in Appendix C discussion of the bus model to predict its behavior under limiting conditions. The prediction was quite accurate and partially validated the model. At lower traffic loads, the demonstration model gave consistently higher delays. The acknowledgements certainly contribute to that higher delay, however no precise prediction was made of that contribution.

Use of Resources. The bus demonstration used a moderate amount of execution time. It ran for twelve 500 msec intervals. Each interval could be considered a simulation run because a data point was generated for each interval. The average execution time per interval was less than seven seconds.

Summary

A wide variety of networks were modeled and demonstrated. Some models presented difficulties which were overcome. However, there is potential for a user to desire protocols that will require development and testing. Such work will be required before validation may be attempted.

The use of resources by the simulation system varies significantly. Parameters such as the node-to-node transmission rate and the type of model significantly influence the use of resources.

VIII. Conclusions and Recommendations

The objective of this thesis was to produce a generic simulation system for computer networks. It was designed using SADT techniques to produce the top levels of the program. The design was implemented using the SLAM combined network-discrete event orientation. The first stage of implementation emphasized the network orientation. As more specific protocols and modeling features were developed, the emphasis shifted to the discrete event orientation. Once implemented, the simulation system was demonstrated.

The last chapter provided an evaluation of the performance of the system in demonstrations of four computer network models. The evaluation was based on the effort required to model the selected networks, validation of the models, and the use of computer resources by the simulation system. Based on those evaluations, several conclusions can be made.

Conclusions

This simulation system is very general. Four very different computer networks were modeled and demonstrated. The user has many specifications available to him in the system. They can be used to build a wide spectrum of computer network models.

Many computer networks may not be fully modeled with the system without further development. Although a wide spectrum of models may be implemented, the specific modeling features

available may not be precise enough to validate the target model.

The simulation system offers useful data for analysis. This analysis can be used to validate, or prove that a model performs according to its specifications. In other words, the simulation model's performance can be mapped precisely to that of a real physical system. Academic inquiries, or investigations that are performed for trade-off analysis only, may still get valuable output from this simulation system without a formal validation as described above.

The use of computer resources by the simulation system may be high, but is quite variable. The parameters of the target model can have a tremendous effect on the amount of execution time required. The execution time required for 500 msec of simulation time ranged from 7 seconds to more than 100 seconds in the demonstrations.

Recommendations

There are three recommendations for individuals interested in using or developing this simulation system. They are based on experience gained in the development of the system and the preceeding conclusions.

The capabilities of the simulation system as offered by this thesis should be carefully analyzed before any use is attempted. Models that may be built with specifications present in the system are prime candidates for simulation using this system. However, if specialized protocols must be developed for

a target model, the user should review three things in order: the design in Chapter III, the SLAM implementation in Chapter V, and the Fortran implementation documentation in Appendix C. A review of the design will identify the points in the current implementation where the new specifications, or protocols, may be interfaced with the rest of the system. The complexity of further development may then be evaluated for a decision on the use of the system.

The implementation of future investigations of this kind that use SLAM should adopt the discrete event orientation. In that orientation, the simulation is started by scheduling events in subroutine INTLC. Subsequent events are scheduled in the event subroutines rather than in the network.

The general nature of this investigation added complexity that made the combined network-discrete event orientation difficult to develop. Keeping track of decisions regarding the overall topology choice adds difficulty when the orientation switches repeatedly. For example, the process that begins the node-to-node transmission and ends in packet reception alternates between orientations several times. The discrete events in the process can make several decisions at different points in time. Decisions based on model type just add to the number of things that the programmer must consider and track. This example covering transmission to reception also illustrates how the complexities of timing can be accentuated when the orientation changes several times. The command of a process that is required by a programmer can slip away when the discrete event

decisions are critically time dependant. When timing is complicated by changes in orientation, the effectiveness of the programmer can be drastically reduced. This difficulty would be eased if the network orientation was dropped.

The overall approach could be made even simpler if the simulation system was broken into multiple sets for mesh, ring, and bus models. I recommend this because most events would not require modification for any of the three models. Of the general events listed in Appendix A, only subroutines QUETOTX and RECVPKT handle the three models differently. QUETOTX is structured like a CASE statement with logical expressions testing the model type. RECVPKT handles packets after testing logical expressions on model type and packet type. Of the flow control and resource handling events, only subroutines FREERSC and OKADMT test the model type and react differently. User functions ten and fifteen are the only user functions that test model type and react differently. Lastly, subroutine INTLC initializes data structures for each model. Most of the changes to these subroutines would not be very difficult. The changes would amount to determining what parts apply to the model desired and removing the parts for the other models. The resulting subroutines would be much clearer.

Testing and debugging would be much easier if the paths taken by messages or packets are not influenced by the type of model chosen. Additionally, the events which are model-specific would not be required in a separate system for the three models. The resulting systems would be more compact and less

demanding of primary memory during execution.

An investigation developing a simulation system without the constraints of the SLAM network or the overhead of three models would enjoy much greater freedom. The investigator would be freer to think and model in any terms he desired. Levels of abstraction such as protocol layers could then be used in the implementation to produce a simulation system design that is conceptually purer and more disciplined.

Bibliography

1. Abrams, Marshall., et al. Computer Networks: A Tutorial. New York: IEEE Press, 1980.
2. Allan, Roger. "Local-net Architecture, Protocol Issues Heating Up," Electronics Design 29 (8): 91-110 (16 April 1981).
3. Bennett, David A. and Christopher A. Landauer. "Simulation of a Distributed System for Performance Modeling," Performance Evaluation Review 8 (3): 49-56 (Fall 1979).
4. Bux, Werner. "Local-Area Subnetworks: A Performance Comparison," IEEE Transactions on Communications C-29 (10): 1465-1471 (October 1981).
5. Christopher, Thomas W. "A Technique for Distributed Simulation of Queueing Systems on Networks of Microcomputers," Proceedings of 2nd Rocky Mountain Symposium on Microcomputers: Systems, Software, Architecture. 402-418. New York: IEEE, 1981.
6. Davies, D.W., et al. Computer Networks and Their Protocols. New York: John Wiley and Sons Ltd., 1979.
7. Elovitz, Honey S. and Constance L. Heitmeyer. "What is a Computer Network?" in Computer Networks: A Tutorial. New York: IEEE Press, 1980.
8. Fortier, Paul J. and Richard G. Leary. "A General Simulation Model for the Evaluation of Distributed Processing Systems," Proceedings of the 14th Annual Simulation Symposium. 215-226. New York: IEEE, 1981.
9. Jafari, Hosein., et al. "Simulation of a Class of Ring-Structured Networks," IEEE Transactions on Computers C-29 (5): 385-391 (May 1980).
10. Kain, R. Y., et al. "CHIMPNET: A Networking Testbed" in 1979 Local Computer Networking. 27-36. New York: IEEE Press, 1979.
11. Pritsker, A. Alan B. and Claude D. Pegden. Introduction to Simulation and SLAM. New York: Halsted Press, 1979.
12. Remes, Antero. "Simulation Techniques in Network Design" in Computer Networks and Simulation, edited by Stephen Schoemaker. New Yor. North-Holland Publishing Company, 1978.

13. Schneider, G. Michael. "VANS -- A Resource-Sharing Computer Network Design Tool" in Computer Networks and Simulation, edited by Stephen Schoemaker. New York: North-Holland Publishing Company, 1978.
14. Schoemaker, S. "On Simulation" in Computer Networks and Simulation. New York: North-Holland Publishing Company, 1978.
15. Schwartz, Mischa. Computer-Communications Network Design and Analysis. New York: Prentice-Hall, Inc., 1978.
16. SofTech, Inc. An Introduction to SADT Structural Analysis and Design Technique. 9022-78R. Waltham, MA: SofTech, 1978.
17. Stewart, Stephen D. Simulation and Analysis of the AFLC Bulk Data Network Using Abstract Data Types. MS Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1981.
18. Thurber, Kenneth. "Special Report: Concise Ethernet Specification," LOCALNetter Newsletter, 2 (4): SR 25-1 - SR 25-4 (Apr 1982).

APPENDIX A

Fortran Subroutine Documentation

The documentation to follow consists of a narrative summary and structured English for each subroutine. Each summary states where the subroutine is called, what it does, and special notes.

The emphasis of the structured English is to help the reader understand what the subroutine does. It was not written as an aid to code reading because some details of the code would obscure the structured English if included. Once a reader understands what the module does, interpreting the code will not be a significant problem if the reader is familiar with Fortran and has reference to the SLAM library routines (9:528-534). The subroutines that this documentation supports are available in the program and documentation package part III.

The presentation to follow is ordered logically. Modules that are related to the same type of network model or same function are grouped together. An overview of the groups, complete with subroutine names and an expansion of each name is given prior to the full presentation of the documentation.

Program and Initialization

Name Expansion

MAIN
EVENT
USERF
INTLC

branch to event subroutines
branch to USERF functions
SLAM initialization

General Events

MAKPKTS
QUETOTX
STOPACT
RETXMT
RECVPKT

make packets
que to transmit
stop activity
retransmit
receive packet

General File Manipulation

PKTPTR
RMVEPKT
ENFILE
COMPLST
MSGDONE
FILEPKT

packet pointer
remove packet
enfile packet
complete list
message done
file packet

Flow Control and Resource Handling

CKADMT	check for admission
ADBRANH	add to node branching
OKADMIT	OK to admit to node
NUMPKTS	number of packets
FREERSC	free resources
NETFREE	free resources from the SLAM Network
FREEBUF	free buffer
SEZEBUF	seize buffer

Mesh Model Specific

MINPATH	minimum path
ROUTER	routing update
INTABLS	initiaize routing tables
PUSH	push on stack
POP	pop off stack
NXTNODE	next node
SVRINDX	server index
HOLDPKT	put packet in hold file
HLDTOUT	hold time-out
STARTTX	start transmission

Ring Model Specific

CONTRL1	controller 1
CONTRL2	controller 2
STPTOKN	stop token
RINGBAK	ring acknowledgment back

Bus Model Specific

BUSTART	start bus server
RESTART	restart bus server
BUSSLLOT	slotted bus for service
CKBUS	check bus for starting
ARBITOR	arbitrator
ARBITOK	arbitration OK
CSMASRT	start bus with CSMA/CD
RXCLSCK	receive collision check
ADDCLSN	add to collision list
CKRXCLN	check for collision at receiving end

Miscellaneous

GNEGNAK	generate negative ACK'S
GENACKS	generate ACK'S (E-E and N-N)
ENTRRTX	enter retransmission process
CLRSVRQ	clear server queue
STATCOL	statistics collection
PRTFILE	print file
PRTSTAT	print statistics
CLRCAL	clear calendar

PROGRAM AND INITIALIZATION

Program Main. This is the driver of the simulation system. It is defined by the SLAM system and requires no user refinements. The only action required is to add user-defined common blocks.

Subroutine EVENT. This is the point of entry into the user-written subroutines from the SLAM network. Each SLAM event node causes execution of the subroutine corresponding to the event code (I).

Function USERF. This is the point of entry into the user-written functions for calls from the SLAM network. Each call to USERF causes execution of the function corresponding to the function code (IFN). There are approximately twenty of these functions. Each has a brief statement of function in the program listing.

Subroutine INTLC. INTLC is called by SLAM to allow any initializations that are required by the user. SLAM network INTLC statements can only make assignments to SLAM variables. Functions performed by INTLC:

- initialize global variables
- initialize the array of completed ID's for each node
- initialize node buffer resources
- initialize data structures for bus networks
- initialize data structures for ring networks
- initialize adjacency list and destination list for mesh networks
- initialize node-to-node distances
- echo selected initializations
- initialize weight table
- execute routing algorithm for each node
- echo results of routing algorithm

SUBROUTINE INTLC

```
(* INITIALIZE GLOBAL SPECIFICATIONS *)  
NUMNODES = GLOBAL SPECIFICATION  
NUMLNS = GLOBAL SPECIFICATION
```

(* INITIALIZE ARRAY FOR ID'S OF MESSAGES THAT HAVE
BEEN COMPLETED *)

FOR ALL NODES

FOR ALL INDEXES

IF (INDEX = POINTER) THEN

DONELIST(INDEX,NODE) = 1

ELSE

DONELIST(INDEX,NODE) = UNDEFINED

END IF

END FOR

END FOR

(* INITIALIZE NODE BUFFERS *)

FOR EACH NODE

BUFFER(NODE) = GLOBAL SPECIFICATION

MAXIMUM(NODE) = GLOBAL SPECIFICATION

END FOR

MINIMUM LEVEL FOR MESSAGE CREATION = GLOBAL SPECIFICATION

MAXIMUM NUMBER OF PACKETS FOR COMMON DESTINATION

AT A NODE = GLOBAL SPECIFICATION

(* INITIALIZE DATA STRUCTURES FOR THE BUS NETWORK *)

FOR ALL NODES

STARTTXTIME(NODE,POINTERINDEX) = 1

ENDTXTIME(NODE,POINTERINDEX) = 1

TRANSMITCOLLISION(NODE) = 0

FOR EACH DESTINATION NODE

STARTTXTIME(NODE,DESTINATION) = -INFINITY

ENDTXTIME(NODE,DESTINATION) = -INFINITY

RXCOLLISION(NODE,DESTINATION,MESSAGEINDEX) = UNDEFINED

RXCOLLISION(NODE,DESTINATION,PACKETINDEX) = UNDEFINED

END FOR

END FOR

FOR ALL NODES

TRANSMITTRY(NODE) = 0

END FOR

NUMBEROFSLOTS = USER SPECIFICATION

(* INITIALIZE DATA STRUCTURES FOR RING NETWORKS *)

SLOT = 1

SLOTSTATE = EMPTY

TOKEN = 1

LASTLENGTH = PACKET OVERHEAD

(* INITIALIZE ADJACENCY LIST AND DESTINATION LIST *)

IF ((MESH NETWORK) .OR. (PROPAGATION DELAY > 0)) THEN

REWIND DESTINATION INFORMATION FILE

END IF

INDEX = 1

IF (MESH NETWORK) THEN

```

FOR EACH NODE
  READ ADJACENT NODES
  FOR EACH LINE
    IF (ADJACENT NODE EXISTS) THEN
      DESTINATION(LINE) = ADJACENT NODE
      ADJACENCY LIST(NODE,LINE) = INDEX
      INDEX = INDEX + 1
    ELSE
      ADJACENCY LIST(NODE,LINE) = 0
    END IF
  END FOR
END FOR
ELSE
  FOR EACH NODE
    FOR EACH LINE
      IF (LINE = 1) THEN
        ADJACENCY LIST(NODE,LINE) = NODE
      ELSE
        ADJACENCY LIST(NODE,LINE) = 0
      END IF
    END FOR
  END FOR
END IF

FOR ALL REMAINING INDEXES
  DESTINATION=0
END FOR

(* READ AND INITIALIZE NODE TO NODE DISTANCES *)
IF (PROPAGATION DELAY.GT.0) THEN
  FOR ALL NODES
    FOR ALL NODES
      READ NODE TO NODE DISTANCE
    END FOR
  END FOR
ELSE
  ALL NODE TO NODE DISTANCES = 0
END IF

(* ECHO RESULTS *)
IF (MESH NETWORK) THEN
  PRINT ADJACENCY LIST
  PRINT DESTINATION LIST
END IF
IF (PROPAGATION DELAY.GT.0) THEN
  PRINT NODE TO NODE DISTANCE TABLE
END IF

IF (MESH NETWORK) THEN
  (* INITIALIZE WEIGHT TABLES *)
  FOR ALL LINES
    FOR ALL NODES
      WEIGHT=1
    END FOR
  END FOR

```

```

(* EXECUTE ROUTING ALGORITHM FOR EACH NODE *)
FOR ALL NODES
  CALL MINPATH
END FOR

(* ECHO RESULTS USING THE NEXT IN TABLE *)
FOR EACH ORIGIN NODE
  FOR EACH DESTINATION NODE
    PRINT NEXTIN NODE
  END FOR
END FOR
END IF
END INTLC

```

GENERAL EVENTS

Subroutine MAKPKTS. Called from the SLAM network when an entity representing a message arrives. The primary function of MAKPKTS is to make several assignments of attributes, packetize the message, and enter each packet into the SLAM network.

Specific functions:

- check node buffer for minimum level. If a minimum available buffer level is not present, suppress the message by returning without entering packets.

- assign length, ID, and packet type

- set error and branching flags

- select destination node and host

- create the packets by assigning lengths and last packet attribute and enter into SLAM network.

NOTE: The packets are created in a somewhat awkward reverse order in order to enter them properly on the SLAM event calendar.

```

SUBROUTINE MAKPKTS
  ASSIGN RESOURCE NUMBER AND QUEUE FROM ATTRIBUTES
  IF (((BUFFER(RESOURCE NUMBER) < MINIMUM) .OR.
    (NUMBERIN(QUEUE)) > MAXIMUM))
    .OR.
    ((NOT MESH MODEL) .AND.
    (NUMBER IN SERVER QUEUE > MAXIMUM QUEUEING NUMBER))) THEN
    (* DO NOT CREATE MESSAGE *)
    IF (MESSAGE CREATION DEPENDS ON PREVIOUS
      MESSAGE ENTRY INTO NODE) THEN
      ENTER INTO SLAM NETWORK FOR TIMING AND BRANCHING
        TO NEXT MESSAGE CREATION
    END IF
    RETURN
  END IF

```

```

PACKETLENGTH = USERF(7)
PACKET ID = COUNTER
INCREMENT(AND RESET)COUNTER
ERROR = FALSE
SERVER BRANCH FLAG = QUEUE
RANDOMLY DETERMINE DESTINATION NODE (ASSIGN TO DESTINATION
AND DATA DESTINATION ATTRIBUTES)
RANDOMLY DETERMINE DESTINATION HOST
(HOST DETERMINATION CURRENTLY ASSUMES TWO HOSTS PER NODE)
PACKET TYPE = DATA
REPEAT
  GET PACKET FROM MESSAGE
  ASSIGN LAST PACKET FLAG
  ENTER PACKET INTO NETWORK NODE
UNTIL (MESSAGE COMPLETELY PACKETIZED)
END MAKPKTS

```

Subroutine QUETOTX. Called from the SLAM network and several other subroutines. The primary function is to take the packet represented by the ATRIB array and enter it into the SLAM network for queueing or for transmission. Several assignments are made prior to entry into the network: next node, server index, and simulated transmission errors.

```

SUBROUTINE QUETOTX
ASSIGN CURRENT NODE AND DESTINATION NODE FROM ATTRIBUTES
IF (MESH NETWORK) THEN
  NEXTNODE = NXTNODE(CURRENT NODE, DESTINATION)
  SERVER = SVRINDEX(CURRENT NODE, NEXTNODE) + 40
ELSE (IF RING NETWORK) THEN
  NEXT = (CURRENT NODE + 1)
  IF (NEXT > NUMNDES) THEN
    NEXTNODE = 1
  ELSE
    NEXTNODE = NEXT
  END IF
  SERVER = CURRENT NODE + 40
ELSE
  NEXTNODE = DESTINATION
  SERVER = CURRENT NODE + 40
END IF

IF (SAMPLE FROM UNIFORM DISTRIBUTION(0.0 TO 1.0)
    < (PROBABILITY OF ERROR) THEN
  ERROR = YES
END IF

IF ((SERVER BRANCH FLAG = QUEUE) .OR.
    (NETWORK IS MESH)) THEN
  ENTER PACKET INTO NETWORK FOR SERVER QUEUEING
ELSE
  ENTER DIRECTLY INTO TRANSMISSION

```

END IF
END QUETOX

Subroutine STOPACT. Called by the SLAM network when a transmission time has expired. It is also scheduled by subroutines that stop transmission earlier than planned. It simply stops the SLAM activity corresponding to the server index. The server index is carried in attribute 13 but must be reduced to less than 50. (This is an accomodation to the SLAM STOPA(NTC) dimension).

```
SUBROUTINE STOPACT
IF (COLLISION WAS NOT ANTICIPATED) THEN
  SERVER INDEX = ATRIB(13) - 30
  END TRANSMISSION OF PACKET IN ACTIVITY FOR SERVER INDEX
END IF
END STOPACT
```

Subroutine RETXMT. Called by the SLAM network upon retransmit time-out and by RECVPKT when a ring acknowledgement contains an error. It looks for the packet in the retransmit file and calls QUETOX if the packet is present.

```
SUBROUTINE RETXMT
IF (PACKET IS IN RETRANSMIT FILE) THEN
  REMOVE IT FROM RETRANSMIT FILE
  ERROR = FALSE
  BRANCH FLAG = QUEUE
  CALL QUETOX
END IF
END RETXMT
```

Subroutine RECVPKT. Called by the SLAM network when a packet completes its transmission activity and the gaining node buffer has room for the packet. RECVPKT seizes the amount of node buffer required for the packet and begins a long series of checks and responses. The checks are in a very precise order to avoid conflicting actions and correctness of response. Errors and collisions in bus networks are handled first, followed by error checks for mesh networks. (Error checking for ring networks is performed where the packet is processed due to the unique acknowledgement scheme for rings.) Then, packets are handled in the following order: node-to-node ACKs, end-to-end ACKs, negative ACKs, ring ACKs, and finally data packets.

```
SUBROUTINE RECVPKT
(ASSIGN VARIABLES FROM ATTRIBUTES)
CURRENT NODE = NEXT NODE
SEIZEBUF(CURRENT NODE,PACKETLENGTH)
```

```

(* FIRST HANDLE BUS NETWORK COLLISIONS AND ERRORS *)
IF (BUS NETWORK) THEN
  IF (COLLISION AT TRANSMITTING NODE) THEN
    RESET COLLISION FLAG
    INCREMENT COLLISION COUNTER(TRANSMIT END)
    IF (RECEIVING END COLLISION) THEN
      INCREMENT COLLISION COUNTER(RECEIVING END)
    END IF
    FREE BUFFER RESOURCE
    RETURN
  ELSE IF (RECEIVING END COLLISION) THEN
    INCREMENT COLLISION COUNTER(RECEIVE END)
    FREE BUFFER RESOURCE
    RETURN
  ELSE IF (ERROR IN TRANSMISSION) THEN
    INCREMENT TRANSMISSION ERROR COUNTER
    FREE BUFFER RESOURCE
    RETURN
  END IF
END IF
(* END SPECIAL BUS HANDLING *)

IF ((ERROR) AND (NETWORK IS MESH) THEN
  INCREMENT ERROR COUNTER
  FREE BUFFER RESOURCE
  IF ((DATA PACKET) AND
      ((NO IMMEDIATE ERROR DETECTION) OR
       (EARLY DETECTION PACKET))) THEN
    CALL GNEGNAK (CURRENT NODE, LAST NODE)
  END IF
  RETURN
END IF

IF (NODE TO NODE ACK) THEN
  FREE BUFFER RESOURCE FOR ACK
  CALL CLRCAL(MSGID,PKTID,CURRENTNODE,RETXMITATTRIBUTE)
  IF (AT DATA ORIGIN NODE) THEN
    CALL HOLDPKT (ORIGIN NODE, PACKET)
  ELSE
    CALL RMVEPKT(RETRANSMIT FILE, PACKET)
    CALL CKADMIT(CURRENT NODE)
  END IF
  RETURN
END IF

IF (END TO END ACK) THEN
  CALL CLRCAL(MSGID,PKTID,CURRENT NODE,RETXMITATTRIBUTE)
  IF (AT DATA ORIGIN NODE) THEN
    FREE BUFFER RESOURCE
    CALL RMVEPKT(RETRANSMIT FILE, PACKET)
    CALL RMVEPKT(HOLD FILE, PACKET)
    CALL CKADMIT(CURRENT NODE)
  ELSE
    CALL QUETOTX

```


END IF
RETURN
END IF

IF (NEGATIVE ACK) THEN
FREE BUFFER RESOURCE FOR ACK
IF (PACKET IN RETRANSMIT FILE) THEN
REMOVE IT FROM RETRANSMIT FILE
CALL QUETOTX
ELSE
CALL ERROR(PACKET NOT IN RETRANSMIT FILE)
END IF
RETURN
END IF

IF (RING ACK) THEN
IF (AT DATA ORIGIN NODE) THEN
FREE RESOURCE FOR ACK
CALL CLRCAL(MSGID,PKTID,CURRENT NODE,RETXMITATTRIBUTE)
IF (NO ERROR) THEN
CALL RMVEPKT(RETRANSITT FILE)
ELSE
CALL RETXMT
INCREMENT ERROR COUNTER
END IF
ELSE
CALL QUETOTX
END IF
CALL CKADMIT(CURRENT NODE)
RETURN
END IF

(* REMAINING PACKETS ARE DATA PACKETS *)
REASSEMBLY FILE = CURRENT NODE + 60
IF (AT DESTINATION NODE) THEN
IF (MESH NETWORK) THEN
CALL ENFILE(REASSEMBLY FILE,PACKET)
CALL GENACKS(LAST NODE,PKTID,CURRENT NODE,
DESTINATION NODE)
ELSE IF (NETWORK IS RING) THEN
IF (NO ERRORS) THEN
CALL ENFILE(REASSEMBLY FILE,PACKET)
ELSE
FREE BUFFER RESOURCE
COLLECT ERROR STATISTICS
END IF
CALL RINGBAK
ELSE IF (NETWORK IS BUS) THEN
CALL ENFILE(REASSEMBLY FILE,PACKET)
CALL GENACKS(LAST NODE,PKTID,CURRENT NODE,
DESTINATION NODE)
RETURN
END IF
ELSE
IF (NETWORK IS RING) THEN

```

        SERVER BRANCH FLAG = TRANSMIT
    END IF
    CALL QUETOTX
    IF (NETWORK IS MESH) THEN
        CALL GENACKS(LASTNODE,PKTID,CURRENT NODE,
                     DESTINATION NODE)
    END IF
END IF
CALL CKADMT(CURRENT NODE)
RETURN
END RECVPKT

```

GENERAL FILE MANIPULATION

Function PKTPTR. Called by several subroutines to find a packet in a file. It steps through the file until the packet is found. The pointer to that packet is returned to the calling subroutine. Special features: If the packet is not found, a zero is returned. If a zero is passed into PKTPTR instead of a packet ID, the pointer to the first packet with a matching message ID is returned.

```

FUNCTION PKTPTR(FILE,PACKET)
NEXT = POINTER TO FIRST ENTRY IN FILE
REPEAT
    IF (NEXT = 0) THEN
        (* EOF IS TRUE *)
        PKTPTR = 0
    ELSE
        COPY ENTRY POINTED TO BY NEXT INTO CHECKARRAY
        IF (ENTRY IN CHECKARRAY = PACKET) THEN
            PKTPTR = NEXT
        ELSE
            NEXT = SUCCESSOR OF ENTRY POINTED TO BY NEXT
        END IF
    END IF
UNTIL ((PACKET FOUND) OR (EOF))

```

Subroutine RMVEPKT. Called by several subroutines. It removes a packet from a file and frees the corresponding node buffer by the packet length.

```

SUBROUTINE RMVEPKT(FILE,MSGID,PKTID)
IF (PACKET ON FILE) THEN
    REMOVE PACKET FROM FILE
    FREE BUFFER RESOURCE
END IF
END RMVEPKT

```

Subroutine ENFILE. Called by RECVPKT. It handles all the filing and processing of data packets. It checks first to see if a packet has the same message ID as any messages that have been previously reassembled and completed. If so, the packet is not filed. The second check is for duplicate packets, or packets that already reside in the reassembly file. If a duplicate is received, it is not filed. Next, a check is made for packets that are complete messages in themselves. They are entered back into the SLAM network for transfer to the destination host. Next, packets are committed to the reassembly processing. This is a complex process that files each packet but stores the current reassembled length of a message in the last packet added to the file. When another packet arrives, the last packet filed is found to check for total length and message completion. When a message is completed, it is entered into the SLAM network for the node-to-host transfer and all packets are removed from the reassembly file.

```
SUBROUTINE ENFILE(FILNUM,PACKET)
  ASSIGN CURRENT NODE, OVERHEAD, MESSAGE SIZE
  ADD LENGTH = PACKET LENGTH - OVERHEAD
```

```
  IF (MESSAGE WAS ALREADY REASSEMBLED) THEN
    FREE BUFFER RESOURCE BY WHOLE PACKET
    RETURN
  END IF
```

```
  IF (PACKET IS ALREADY IN REASSEMBLY FILE) THEN
    FREE BUFFER RESOURCE BY WHOLE PACKET
    (* DISCARD DUPLICATE *)
    RETURN
  END IF
```

```
  COLLECT PACKET DELAY STATISTICS
```

```
  IF (ADD LENGTH = MESSAGE SIZE) THEN
    FREE BUFFER RESOURCE BY OVERHEAD
    ENTER INTO NETWORK FOR NODE-HOST TRANSFER
    CALL COMPLST(CURRENT NODE,MSGID)
    RETURN
  END IF
```

```
  RSMPTR = LAST ENTRY IN REASSEMBLY FILE
  REPEAT
```

```
    IF (FILE EMPTY) THEN
      FILE PACKET IN REASSEMBLY FILE
    ELSE
      COPY PACKET POINTED TO BY RSMPTR INTO CHECK ARRAY
      IF (PACKET FROM FILE HAS SAME MSGID) THEN
        IF (REASSEMBLED MESSAGE LENGTH = MESSAGE SIZE) THEN
          FREE BUFFER RESOURCE BY OVERHEAD OF PACKET AND
            OVERHEAD OF REASSEMBLED MESSAGE
```

```

ENTER INTO NETWORK FOR NODE-HOST TRANSFER
CALL COMPLST(CURRENNT NODE,MSGID)
REPEAT
  REMOVE PACKET OF MESSAGE FROM FILE
  UNTIL (ALL PACKETS REMOVED)
ELSE
  ADD PACKET LENGTH TO PARTIALLY REASSEMBLED MESSAGE
  FILE MESSAGE IN REASSEMBLY FILE
  FREE BUFFER RESOURCE BY PACKET OVERHEAD
END IF
ELSE
  RSMPTR = POINTER TO PREDESSOR OF RSMPTR
END IF
END IF
UNTIL ((PACKET FILED) OR (NODE-HOST TRANSFER STARTED))
END ENFILE

```

Subroutine COMPLST. Called by ENFILE when a message is complete. It enters the ID of a completed message into an array. It operates like an infinite circular stack to keep the last twenty five message ID's that were completed. Note: Subroutines COMPLST and MSGDONE are the only means of accessing the array of completed messages.

```

SUBROUTINE COMPLST(NODE,MSGID)
INDEX = DONELIST(POINTER,NODE)
DONELIST(INDEX,NODE) = MSGID
INCREMENT INDEX
IF (INDEX.GT.DONELIST DIMENSION) THEN
  DONELIST(POINTER,NODE) = 1
ELSE
  DONELIST(POINTER,NODE) = INDEX
END IF
END COMPLST

```

Function MSGDONE. Called by ENFILE. It checks the array of completed message ID's for a particular ID. It returns a truth value based on whether there was a match.

```

LOGICAL FUNCTION MSGDONE(NODE,MSGID)
MSGDONE = FALSE
INDEX = 1
REPEAT
  IF (DONELIST(INDEX,NODE).EQ.MSGID) THEN
    MSGDONE = TRUE
  ELSE
    INCREMENT INDEX
  END IF
UNTIL (ENTIRE ARRAY IS CHECKED)
END MSGDONE

```

Subroutine FILEPKT. called by the SLAM network after a packet is processed by a node and is ready to be queued for a server. FILEPKT files the packet in the SLAM file representing the appropriate server queue.

```
SUBROUTINE FILEPKT
  ASSIGN FILE NUMBER FROM ATTRIBUTE
  CALL FILEM(FILENUMBER)
END FILEPKT
```

FLOW CONTROL AND RESOURCE HANDLING

Subroutine CKADMT. Called by several subroutines after the node buffer has been freed in a routine. If a packet is present in the AWAIT node for the node, and the criteria for entry into the node is met, the packet is admitted into the node. Several packets can be entered by CKADMT if they meet the criteria for entry.

```
SUBROUTINE CKADMT(NODE)
  FILE= NODE+20
  REPEAT
    IF (FILE NOT EMPTY) THEN
      COPY PACKET LENGTH, PACKETTYPE, NODE,
        AND DESTINATION FROM ATTRIBUTES OF PACKET ON FILE
      IF (OKADMIT(ARGUMENT LIST)) THEN
        REMOVE FIRST ENTRY OF FILE
        CALL SEZEBUF(NODE, PACKET LENGTH)
        ENTER PACKET INTO SLAM NETWORK
        IF (MESSAGE CREATION DEPENDS ON PREVIOUS MESSAGE
          ENTRY INTO NODE) THEN
          ENTER NETWORK FOR BRANCHING TO RECREATIONS
        END IF
      END IF
    END IF
  UNTIL ((FILE IS EMPTY).OR.(.NOT. OKADMIT))
END CKADMT
```

Subroutine ADBRANH. Called by the SLAM network when a packet is queued for admission to a node. A decision based on the node buffer (and other constraints that are implemented in subroutine OKADMT) is made and a branching flag is set. Based on that flag, the SLAM network will leave the packet queued in the host, or admit the packet into the network.

```
SUBROUTINE ADBRANH
  ASSIGN PACKETLENGTH,PACKETYPE,CURRENT AND DESTINATION
    NODES FROM ATTRIBUTES
  IF (OKADMIT(ARGUMENT LIST)) THEN
    CALL SEZEBUF (NODE,PACKET LENGTH)
    BRANCH SWITCH = SEND TO NODE
```

```

      IF (MESSAGE CREATION DEPENDS ON PREVIOUS MESSAGE
          ENTRY INTO NODE) THEN
        ENTER INTO SLAM NETWORK FOR TIMING AND BRANCHING
          TO NEXT CREATION
      END IF
    ELSE
      BRANCH SWITCH = SEND TO HOST QUEUE
    END IF
  END ADBRANH

```

Function OKADMIT. Called by subroutines ADBRANH and CKADMT. This module tests the criteria for packet entry into the node. The number of packets that have the same destination that have already been admitted to the node is checked against a maximum number. Also, the amount of buffer resource that would be left after the packet admission is checked against a minimum. OKADMIT is currently set up to allow acknowledgement packets to enter based on the buffer capacity alone.

```

FUNCTION OKADMIT (NODE,PACKET LENGTH,DESTINATION)
(MINIMUM LEVEL AND MAXPACKETS ARE GLOBALLY SPECIFIED)

IF (DATA PACKET) THEN
  TOTPACKETS = NUMPKTS (HOLD FILE, DESTINATION)
    + NUMPKTS (RETRANSMITFILE, DESTINATION)
  FOR EACH LINE
    INDEX = ADJACENCY LIST(NODE,LINE)
    SERVER FILE = INDEX + 40
    IF (SERVER FILE EXISTS) THEN
      TOTPKTS = TOTPKTS + NUMPKTS(SERVER FILE,DESTINATION)
    END IF
  END FOR
  BUFFER LEVEL = CURRENT BUFFER (NODE) - MINIMUM LEVEL
ELSE
  TOTPKTS = 0
  BUFFERLEVEL = CURRENT BUFFER (NODE)
END IF
IF (MESH MODEL) THEN
  OKADMIT = ((TOTPACKETS <= MAXPACKETS).AND.
    (PKTLENGTH <= BUFFERLEVEL))
ELSE
  SERVERFILE = NODE + 40
  OKADMIT = (((TOTPACKETS <= MAXPACKETS).AND.
    (PKTLENGTH <= BUFFERLEVEL)))
    .AND.
    (NUMBER IN SERVERFILE < MINIMUM QUEUING NUMBER))
END OKADMIT

```

Function NUMPKTS. Called by subroutine OKADMIT. It returns the number of packets in a file that have the same destination.

```

FUNCTION NUMPKTS (FILE,DEST)
COUNT = 0

```

```

    POINTER = FIRST ENTRY OF FILE
    WHILE (NOT END OF FILE) DO
        CKECKDEST = ATTRIBUTE 5 OF POINTER ENTRY
        IF (CKECKDEST = DEST) INCREMENT COUNT
        POINTER = SUCCESSOR OF POINTER
    END WHILE
    NUMPKTS = COUNT
END NUMPKTS

```

Subroutine FREERSC. Called by the SLAM network when a packet ends transmission. For ring models, It frees the transmitting node buffer by the length of the packet only. For mesh and bus models, it clears the server queue and handles the decision for retransmission as well as buffer handling.

```

SUBROUTINE FREERSC
ASSIGN VARIABLES FROM ATTRIBUTES
IF (NETWORK = RING) THEN
    IF (DATA DESTINATION IS NOT LASTNODE) .AND.
        (LASTNODE IS NOT ORIGIN NODE)) THEN
        FREE BUFFER AT LASTNODE BY PACKETLENGTH
    END IF
    ELSE IF (THERE WAS NO COLLISION AT TRANSMITTING NODE) THEN
        CALL CLRSVRQ(LASTNODE,SERVER FILE,MSGID,PKTID)
    END IF
CALL CKADMT(LASTNODE)
END FREERSC

```

Subroutine NETFREE. Called from the SLAM network when a message transfers from a node to a host. It frees the node buffer by the length of the message and checks for packets awaiting entry.

```

SUBROUTINE NETFREE
ASSIGN AMOUNT AND NODE FROM ENTITY ATTRIBUTES
CALL FREEBUF (NODE,AMOUNT)
CALL CKADMT(NODE)
END NETFREE

```

Subroutine FREEBUF. Called by several different subroutines to free node buffer resources. It performs complete error checking on the arguments and buffer capacity prior to freeing the buffer. NOTE: Subroutines FREEBUF and SEZEBUF are the only means of changing the buffer resources utilization after initialization.

```

SUBROUTINE FREEBUF (NODE,AMOUNT)
IF ((NODE < 1).OR.(NODE > NUMNDES)) THEN
    CALL ERROR(NODE ARGUMENT OUT OF RANGE)
END IF

IF (AMOUNT <= 0) THEN

```

```

    CALL ERROR(ATTEMPT TO FREE NEGATIVE RESOURCES)
END IF

```

```

IF ((BUFFER(NODE) + AMOUNT) > MAXIMUM(NODE)) THEN
    CALL ERROR(ATTEMPT TO FREE BEYOND CAPACITY)
ELSE
    BUFFER(NODE) = BUFFER(NODE) + AMOUNT
END IF
END FREEBUF

```

Subroutine SEZEBUF. Called by several subroutines to seize node buffer resources. It performs complete error checking on the arguments and buffer capacity prior to seizing the buffer.

```

SUBROUTINE SEZEBUF (NODE,AMOUNT)
IF ((NODE < 1).OR.(NODE > NUMNDES)) THEN
    CALL ERROR(NODE ARGUMENT OUT OF RANGE)
END IF

IF (AMOUNT <= 0) THEN
    CALL ERROR(ATTEMPT TO SEIZE NEGATIVE RESOURCES)
END IF

IF (BUFFER(NODE)-AMOUNT) < 0) THEN
    CALL ERROR(ATTEMPT TO SEIZE BETWEEN CAPACITY)
ELSE
    BUFFER(NODE) = BUFFER(NODE)-AMOUNT
END IF
END SEZEBUF

```

MESH MODEL SPECIFIC

Subroutine MINPATH. Called by subroutines INTLC and ROUTER. MINPATH is the heart of the routing algorithm used for models of distributed topology. It uses a minimum path algorithm to produce for each node a next in table. The next in table is used to determine the next node to for each destination. For additional information on this implementation or data structures see Chapter IV.

```

MINPATH (ORIGIN NODE)
    INITIALIZE DISTANCE TABLE (ORIGIN)
    INITIALIZE NEXT IN TABLE (ORIGIN)
    INITIALIZE STACKS
    STACK INDEX=0
    FOR EACH DESTINATION FROM ORIGIN
        DISTANCE(ORIGIN,DESTINATION) = DISTANCE(ORIGIN,ORIGIN)
        + WEIGHT(ORIGIN,DESTINATION)
        PUSH DESTINATION ON STACK(STACKINDEX)
        NEXTIN(ORIGIN,DESTINATION) = ORIGIN
    END FOR

```



```

END FOR
STACKINDEX = 2
REPEAT
  POP NODE FROM STACK(STACKINDEX MOD 2)
  FOR EACH DESTINATION FROM THAT NODE
    IF (DISTANCE(ORIGIN,DESTINATION) >
        DISTANCE(ORIGIN,NODE) + WEIGHT(NODE,DESTINATION)
    THEN
      DISTANCE(ORIGIN,DESTINATION)=DISTANCE(ORIGIN,NODE)
        + WEIGHT(NODE,DESTINATION)
      PUSH(DESTINATION) ON STACK((STACKINDEX+1)MOD 2)
      NEXTIN(ORIGIN,DESTINATION) = NODE
    END IF
  END FOR
  IF STACK(STACKINDEX MOD 2) IS EMPTY THEN
    STACKINDEX = STACKINDEX +1
  END IF
UNTIL (STACK(STACKINDEX MOD 2)) IS EMPTY
END MINPATH

```

SUBROUTINE ROUTER. Called from the SLAM network at each routing table update time period. It updates the routing weight table for a node based on the number of packets in each server queue. Then, it executes the minimum path routing algorithm for the node. Each nodes routing tables are updated and results stored in the NEXTIN table.

```

SUBROUTINE ROUTER
FOR EACH NODE
  FOR EACH LINE
    SERVER INDEX = ADJACENCY LIST (NODE,LINE)
    IF (SERVER INDEX EXISTS) THEN
      SERVERQUE = INDEX + 40
      WEIGHT(INDEX,NODE) = (LENGTH(SERVERQUE)
        * WEIGHTING FACTOR) + 1
    END IF
  END FOR
  CALL MINPATH(NODE)
END FOR
END ROUTER

```

Subroutine INTABLS. Called only by MINPATH. INTABLS initializes the DISTANCE table and NEXTIN table for the node passed as an argument. It also initializes the stacks.

```

SUBROUTINE INTABLS(ORIGIN NODE)
FOR ALL NODES
  IF (NODE IS ORIGIN) THEN
    DISTANCE(ORIGIN,NODE) = 0
  ELSE
    DISTANCE(ORIGIN,DESTINATION) = INFINITY
  END FOR

```

```

END FOR
FOR EACH DESTINATION
    NEXTIN(ORIGIN,DESTINATION) = 0
END FOR
STACK(0,POINTER)=0
STACK(1,POINTER)=0
RETURN
END INTABLS

```

Subroutine PUSH. Called only by MINPATH. PUSH selects the stack indicated by the index, and stores the entry onto that stack.

```

SUBROUTINE PUSH(INDEX,ENTRY)
IF (STACK(POINTER,INDEX) = NUMNDES) then
    CALL ERROR(STACK OVERFLOW)
END IF
NEXT SLOT = STACK(INDEX)POINTER + 1
STACK(INDEX,NEXTSLOT) = ENTRY
STACK(INDEX,POINTER) = NEXTSLOT
END PUSH

```

Function POP. Called only by MINPATH. POP selects the stack indicated by the index and returns the top entry from that stack.

```

FUNCTION POP(INDEX)
POPSLOT = STACK(POINTER,INDEX)
IF (POPSLOT <= 0) CALL ERROR(POPPING EMPTY STACK)
POP = STACK(POPSLOT,INDEX)
STACK(POINTER,INDEX) = POPSLLOT - 1
END POP

```

Function NXTNODE. Called by QUETOTX. NXTNODE returns the next node that should be taken by the origin node to reach the destination node. NXTNODE is used only in mesh models.

```

SUBROUTINE NXTNODE (ORIGIN,DESTINATION)
NEXT = DESTINATION
REPEAT
    IF(NEXTIN FROM NEXT = ORIGIN) THEN
        NXTNODE = NEXT
    ELSE (IF THERE IS NO PATH)
        PRINT WARINING
    ELSE
        NEXT = NEXTIN FROM NEXT
UNTIL ((NXTNODE IS FOUND) OR (NO PATH EXISTS))
END NXTNODE

```

Function SVRINDX. Called by QUETOTX. SVRINDX returns the index of the line between current node and next node. The index is the means of selection of the server for mesh models.

```
SUBROUTINE SVRINDX (CURRENT NODE, NEXT NODE)
LINE = 1
REPEAT
INDEX = ADJACENCY LIST(CURRENT NODE,LINE)
  IF (DESTINATION(INDEX) = NEXTNODE) THEN
    SVRINDX = INDEX
  ELSE
    INCREMENT LINE
    IF LINE > NUMLNS CALL ERROR(EXCEEDED NUMBER OF LINES)
  END IF
UNTIL (SVRINDX FOUND)
END SVRINDX
```

Subroutine HOLDPKT. Called by the SLAM network when a node-to-node ACK is received at the originating node. It removes the packet from the retransmit file and enters it into the SLAM network for hold queueing and time-out. Used only in mesh models.

```
SUBROUTINE HOLDPKT
GET POINTER TO PACKET
IF (PACKET IS IN RETRANSMITT FILE) THEN
  REMOVE IT FROM RETRANSMITT FILE
  ENTER IT INTO HOLD FILE
END IF
END HOLDPKT
```

Subroutine HLDTOU. Called from the SLAM network when a packet has timed out from the hold queue. If the packet is still present, it is removed and placed back into the appropriate server queue. Used only in mesh models.

```
SUBROUTINE HLDTOU (* HOLD TIME-OUT *)
IF (PACKET IS IN HOLD FILE) THEN
  REMOVE IT FROM HOLD FILE
  CALL QUETOTX
END IF
END HLDOUT
```

Subroutine STARTTX. Called by the SLAM network for mesh models only. It is called when a packet enters a server queue and when a packet ends transmission. In both cases, it checks for a packet on queue and an idle server prior to committing the first packet on queue to transmission.

```
SUBROUTINE STARTTX
ASSIGN SERVER FROM ATTRIBUTES
```

```

IF ((SERVER IDLE).AND.(SERVER QUEUE.NOT EMPTY)) THEN
  COPY FIRST PACKET FROM SERVER QUEUE
  ENTER IT DIRECTLY INTO SLAM NETWORK FOR TRANSMISSION
END IF
END STARTTX

```

RING MODEL SPECIFIC

Subroutine CONTRL1. Called by the SLAM network for ring models using a token network access scheme. It models the token ring of Bux(4). It is coupled closely to the SLAM network through several attributes. These attributes must be assigned for use by SLAM network user functions for propagation delay and transmission time. LASTORIGIN is used to store the last node that originated a packet, a key feature to prevent a node seizing the ring for all of its packets.

```

SUBROUTINE CONTRL1
SERVER QUEUE = QUEUE FOR NODE WHERE TOKEN IS
ASSIGN LASTORIGIN FROM ATTRIBUTE
ASSIGN ATTRIBUTES FOR THE TOKEN ENTITY THAT REPRESENT
                                ORIGIN, CURRENT, AND NEXT NODE

IF ((SERVER QUEUE NOT EMPTY).AND.(TOKEN NOT AT LASTORIGIN))
                                THEN
  IF (SERVER QUEUE NOT EMPTY) THEN
    REMOVE FIRST PACKET
    TOKEN ATTRIBUTE FOR LASTORIGIN = NODE WHERE TOKEN IS
    ENTER PACKET INTO SLAM NETWORK FOR TRANSMISSION
    LASTLENGTH = PACKET LENGTH
  ELSE
    LASTLENGTH = PACKET OVERHEAD
  END IF
ELSE
  LASTLENGTH = RING LATENCY
END IF

IF (AT NODE THAT LAST ORIGINATED A PACKET) THEN
  TOKEN ATTRIBUTE FOR LASTORIGIN = UNDEFINED
END IF

TOKEN ATTRIBUTE FOR LENGTH = LASTLENGTH
ENTER TOKEN ENTITY INTO SLAM NETWORK
MOVE TOKEN TO NEXT NODE
END CONTRL1

```

Subroutine CONTRL2. Called by the SLAM network for ring models using a slotted network access scheme. CONTRL2 models the slotted ring of Bux(4). This implementation is characterized by a SLAM controller loop that is synchronized (coupled closely)

with the transmission process. As a result, the controller loop requires attributes for the SLOT entity for controlling propagation delay and transmission time. Reference to attributes are to those of the SLOT entity.

```
SUBROUTINE CONTRL2
ASSIGN LASTORIGIN AND SLOTYPE FROM ATTRIBUTES
NODE = SLOT
IF (SLOTYPE = FULL) THEN
  IF (NODE = LASTORG) THEN

    SLOTYPE ATTRIBUTE = JUST EMPTIED
    LASTORIGIN ATTRIBUTE = UNDEFINED
  END IF
ELSE IF (SLOTYPE = JUST EMPTIED) THEN

  SLOTYPE ATTRIBUTE = EMPTY
END IF
SERVER = NODE + 40
IF (((SERVER IS IDLE).AND.(SERVER QUEUE NOT EMPTY))
    .AND.(SLOTYPE ATTRIBUTE = EMPTY)) THEN
  REMOVE FIRST PACKET FROM SERVER QUEUE
  SLOTYPE ATTRIBUTE = FULL
  LASTORIGIN ATTRIBUTE = SLOT
  ENTER PACKET INTO SLAM NETWORK FOR TRANSMISSION
END IF

CURRENT NODE ATTRIBUTE = SLOT
INCREMENT SLOT LOCATION
NEXT NODE ATTRIBUTE = SLOT
ENTER SLOT ENTITY INTO SLAM SLOTTED RING CONTROLLER LOOP
END CONTRL2
```

Subroutine STPTOKN. Called by the SLAM network. It is called when the packet just released by the node with the token arrives at the next node.

```
SUBROUTINE STPTOKN
STOP TOKEN RING CONTROLLER CYCLE
END STPTOKN
```

Subroutine RINGBAK. Called by RECVPKT when a packet in a ring model reaches its destination. It generates a ring ACK and enters it into the SLAM network for immediate transmission to the next node.

```
SUBROUTINE RINGBAK
PACKET TYPE = RING ACK
DESTINATION = DATA ORIGIN
SERVER BRANCH FLAG = TRANSMIT
CALL QUETOTX
END RINGBAK
```

BUS MODEL SPECIFIC

Subroutine BUSTART. Called by the SLAM network for BUS models when a packet enters the server queue and it is the only one present. It calls for arbitration of the bus. If arbitration is successful, subroutine CHKBUS is called to begin the transmission process. otherwise, CHKBUS is scheduled at the future arbitrated time. CHKBUS is never scheduled for a future time with the current subroutine ARBITOR. This feature is included to allow users to modify arbitor without modification to subroutines BUSTART or RESTART which also allows future scheduling of CHKBUS.

```
SUBROUTINE BUSTART
ASSIGN NODE FROM ATTRIBUTE
CALL ARBITOR
IF (ARBITOK(NODE)) THEN
    CALL CHKBUS
ELSE IF (ENDOK TIME(NODE) > TNOW) THEN
    CALL SCHEDULE(CHKBUS AT ENDOK TIME(NODE))
END IF
END BUSTART
```

Subroutine RESTART. Called by the SLAM network for bus models when a server has ended transmission of a packet. It clears the last packet from the server queue if there was no collision at the transmission end. If another packet is still present on the server queue, RESTART will call for arbitration and either call CHKBUS or schedule CHKBUS to begin the transmission process.

```
SUBROUTINE RESTART
ASSIGN VARIABLES FROM ATTRIBUTES
IF (SLOTTED MODEL) RETURN
IF (COLLISION AT TRANSMITTING NODE) THEN
    TRANSMITTRY(NODE) = TRANSMITTRY(NODE) + 1
    CALL SCHEDULE(CHKBUS AFTER RANDOM BACKOFF DELAY)
ELSE
    TRANSMITTRY(NODE) = 0
    IF ((THERE IS ANOTHER PACKET IN SERVER QUEUE) THEN
        CALL ARBITOR
        IF (ARBITOK(NODE)) THEN
            GET NEXT PACKET
            CALL SCHEDULE(CHKBUS AFTER MINIMUM PACKET SPACING)
        ELSE IF (ENDOK TIME(NODE) > TNOW) THEN
            GET NEXT PACKET
            CALL SCHEDULE(CHKBUS AT ENDOK TIME(NODE))
        END IF
    END IF
END IF
END RESTART
```

Subroutine BUSSLOT. Called by the SLAM network on a cyclic basis. It provides the control for the slotted bus (slotted ALOHA). For each slot, it performs the arbitration function for each node and commits packets to transmission if they are present. The SLAM ATRIB array is protected against global interference.

```

SUBROUTINE BUSSLOT
SAVE SLAM ATRIB ARRAY
FOR ALL NODES
  (*RE-ARBITRATE*)
  STARTOK TIME(NODE) = TNOW
  ENDOK TIME(NODE) = TNOW
  (*START PACKET*)
  SERVER FILE = NODE + 40
  IF (SERVER FILE NOT EMPTY) THEN
    GET THE FIRST PACKET
    CALL CHKBUS
  END IF
END FOR
RESTORE SLAM ATRIB ARRAY
END BUSSLOT

```

Subroutine CHKBUS. Called by subroutines BUSSTART, RESTART, and BUSSLOT. Also scheduled by BUSTART and RESTART. It begins the transmission process via subroutine CSMAST for CSMA/CD models. Otherwise, it begins the transmission process directly by entering a copy of the first packet into the SLAM network and assigning the node times for start and end of transmission.

```

SUBROUTINE CHKBUS
ASSIGN VARIABLES FROM ATTRIBUTES
IF (CSMA/CD SPECIFIED) THEN
  CALL CSMAST
ELSE
  ENTER PACKET INTO TRANSMISSION
  NEXTSTARTSLOT = STRTIME(NODE, POINTERINDEX) + 1
  IF (NEXTSTARTSLOT > NUMBEROFSLOTS) NEXTSTARTSLOT = 1
  STRTIME(NODE, POINTERINDEX) = NEXTSTARTSLOT
  STRTIME(NODE, NEXTSTARTSLOT) = TNOW
  IF (RANDOM DELAY MODEL) THEN
    ENDTIME(NODE, NEXTSTARTSLOT) = TNOW + PACKET TX TIME
  ELSE
    ENDTIME(NODE, NEXTSTARTSLOT) = TNOW + BUS SLOT LENGTH
  END IF
  ASSIGN THE FOLLOWING VARIABLES FROM ATTRIBUTES:
    TXIDS(NODE, MESSAGEINDEX)
    TXIDS(NODE, PACKETINDEX)
    TXIDS(NODE, DESTINATIONINDEX)
  CALL RXCLSCK
END IF
END CHKBUS

```

Subroutine ARBITOR. Called by BUSTART and RESTART to arbitrate for access to the bus for a node. ARBITOR presently contains no future timing using the random delay access scheme. Other access schemes can be inserted by using a case statement structure or by replacing ARBITOR. The result of arbitration is stored in a start and end time that the node has arbitrated over the bus. Arbitration schemes that work on a cycle, or synchronously, are implemented more easily in a SLAM network timing loop as was done using BUSSLOT.

```
SUBROUTINE ARBITOR
IF (RANDOM DELAY ARBITRATION) THEN
  ASSIGN NODE FROM ATTRIBUTE
  STRTOK(NODE) = TNOW
  ENDOK(NODE) = STRTOK(NODE)
END IF
END ARBITOR
```

Function ARBITOK. Called by subroutines BUSTART and RESTART. For a given node, it returns a truth value that indicates if TNOW is within the arbitrated time.

```
LOGICAL FUNCTION ARBITOK(NODE)
ARBITOK = (TNOW >= STRTOK(NODE))
          AND
          (TNOW <= ENDOK(NODE))
END ARBITOK
```

Subroutine CSMAST. Called by CHKBUS when CSMA/CD is specified for the model. For each node other than the one that is considering accessing the bus, the signal arrival times are checked to determine the response of the node seeking access. The node may sense a signal and thus defer access by rescheduling CSMAST. If the bus is inactive, the node will begin transmission. However, if during the signal checking for each node a future overlap will occur, the collisions are scheduled and flag are set. The collisions are implemented by scheduling the transmission process stop event. The collision flag is used by RECVPKT to ignore the corrupted packet.

```
SUBROUTINE CSMAST
COLLISIONTIME = INFINITY
ENDTXHERE = TNOW + TRANSMISSION TIME OF PACKET
OKTOSTART = TRUE
COLLISIONPOTENTIAL = FALSE
(*ASSIGN ORGNODE AND DESTNODE FROM ATTRIBUTES*)
RESET STOPA DISABLE FLAG
```

```
FOR EACH NODE
  IF (NOT ORGNODE) THEN
    FOR EACH SLOT (OF TRANSMISSION SPACE)
```



```

TXARVL = STRTIME(NODE,SLOT) + PROPD(ORGNODE,NODE)
TXEND = ENDTIME(NODE,SLOT) + PROPD(ORGNODE,NODE)
IF ((TNOW.GT.TXARVL).AND.(TNOW.LT.TXEND)) THEN
  (*BUS ACTIVE*)
  OKTOSTART = FALSE
ELSE IF ((TNOW.LT.TXARVL).AND.
          (ENDTXHERE.GT.TXARVL)) THEN
  (*COLLISION WILL OCCUR*)
  COLLISIONTIME = MINIMUM(COLLISIONTIME,TXARVL-TNOW)
  COLLISIONPOTENTIAL = TRUE
END IF
END FOR
END IF
END FOR

IF (OKTOSRT) THEN
  IF (COLLISIONPOTENTIAL) THEN
    STOPA DISABLE FLAG = DISABLE
  END IF
  ENTER INTO SLAM NETWORK FOR TRANSMISSION
  NEXTSTARTSLOT = STRTIME(NODE,POINTERINDEX) + 1
  IF (NEXTSTARTSLOT > NUMBEROFSLOTS) NEXTSTARTSLOT = 1
  STRTIME(ORGNODE,POINTERINDEX) = TNOW
  ENDTIME(ORGNODE,NEXTSTARTSLOT) = TNOW +
    MINIMUM (TRANSMIT TIME, COLLISION TIME)
  TXIDS(ORGNODE,MESSAGEINDEX) = MSGID
  TXIDS(ORGNODE,PACKETINDEX) = PKTID
  TXIDS(ORGNODE,DESTINATIONINDEX) = DESTINATION NODE

  FOR ALL NODES
    NODESLOT = STRTIME(NODE,POINTERINDEX)
    IF (NODE <> ORGNODE) THEN
      TXARVL = STARTIME(ORGNODE,NEXTSTARTSLOT) +
        PROPD(ORGNODE,NODE)
      IF (TXARVL < ENDTIME(NODE,NODESLOT)) THEN
        CALL CLRCAL(MSGID,PKTID,NODE,MARKSTOPATRIB)
        IF (NO TXCOLLISION AT NODE) THEN
          TXCOLLISION(NODE) = TXCOLLISION(NODE) + 1
        END IF
        ENDTIME(NODE,NODESLOT) = TXARVL
        TXCLASSN(NODE) = TRUE
        ASSIGN THE FOLLOWING ATTRIBUTES TO AN ENTITY
          THAT WILL STOP TRANSMISSION FOR THE MESSAGE
        MSGID ATTRIBUTE = TXIDS(NODE,MESSAGEINDEX)
        PKTID ATTRIBUTE = TXIDS(NODE,PAK CETINDEX)
        SERVER INDEX ATTRIBUTE = NODE + 40
        DESTINATION ATTRIBUTE =TXIDS(NODE,DESTINATIONINDEX)
        STOPA DISABLE FLAG = ENABLED
        CALL SCHEDULE (STOPACT AT TXARVL USING ABOVE
          ATTRIBUTES)
      END IF
    END IF
  END FOR
  CALL RXCLSSK

```

```

ELSE IF (NOT SLOTTED BUS MODEL) THEN
  TRANSMITTRY(ORGNODE) = TRANSMITTRY(ORGNODE) + 1
  CALL SCHEDULE(CSMASRT AFTER RANDOM BACKOFF DELAY)
END IF
IF ((COLLISIONPOTENTIAL).AND.(OKTOSRT)) THEN
  TXCOLLISION(ORGNODE) = TXCOLLISION(ORGNODE) + 1
  STOPA DISABLE FLAG = ENABLED
  CALL SCHEDULE (STOPACT AT COLLISION TIME)
END IF
END CSMARST

```

Subroutine RXCLSCK. Called by CHKBUS and CSMASRT when a packet is committed to transmission. For each node, the signals from all other nodes and the signal from the node that just started transmitting are checked for overlap. If any signals overlap, a collision is recorded for the triple (transmitting node,message ID, and packet ID) with a call to addclsn.

```

SUBROUTINE RXCLSCK
FOR EACH ORGNODE
  LASTSTART = STRTIME(ORGNODE, POINTERINDEX)
  IF ((ENDTIME(ORGNODE, LASTSTART) + MAX PROPAGATION DELAY >
                                           TNOW) THEN
    ORGDEST = TXIDS(ORGNODE, DESTINATIONINDEX)
    MSGID = TXIDS(ORGNODE, MESSAGEINDEX)
    PKTID = TXIDS(ORGNODE, PACKETINDEX)
    PKTARVL = STRTIME(ORGNODE) + PROPDELAY(ORGNODE, DESTNODE)
    PKTEND = ENDTIME(ORGNODE) + PROPDELAY(ORGNODE, DESTNODE)

    FOR EACH NODE
      LASTSTART = STRTIME(NODE, POINTERINDEX)
      IF ((NODE.NE.ORGNODE) .AND.
          (ENDTIME(NODE, LASTSTART) + MAX PROPAGATION DELAY >
                                           TNOW)) THEN
        FOR EACH SLOT
          SIGARVL = STRTIME(NODE, SLOT) +
                    PROPDELAY(NODE, DESTNODE)
          SIGEND = ENDTIME(NODE, SLOT) +
                    PROPDELAY(NODE, DESTNODE)
          IF((((SIGARVL.GE.PKTARVL).AND.
                (SIGARVL.LT.PKTEND)))
              .OR.
              ((SIGEND.GT.PKTARVL).AND.(SIGARVL.LE.PKTEND)))
              .OR.
              (SIGARVL.LT.PKTARVL).AND.(SIGEND .GT.PKTEND))))
            THEN
              COLLISION = TRUE
            END IF
          END FOR
        END IF
      END FOR
    END IF
  END FOR

```

Subroutine ADDCLSN. Called by RXCLSCK when it is determined that a message will be corrupted by a collision at the receiving node. ADDCLSN adds the message and packet ID's to a list of corrupted packets. There is a list of such packets stored for each node in array RXCOLLISION. Array RXCOLLISION is accessed only by ADDCLSN and function CKRXCLN. ID's are added to RXCOLLISION in an infinite circular stack fashion.

```
SUBROUTINE ADDCLSN(NODE,MSGID,PKTID)
  SLOT = RXCOLLISION(NODE,INDEX,INDEX)
  RXCOLLISION(NODE,SLOT,MESSAGEINDEX) = MSGID
  RXCOLLISION(NODE,SLOT,PACKETINDEX) = PKTID
  SLOT = NEXT CIRCULAR SLOT
  RXCOLLISION(NODE,INDEX,INDEX) = SLOT
END ADDCLSN
```

Function CKRXCLN. Called by RECVPKT for bus models. It checks the RXCOLLISION array to see if the received message was corrupted by a collision at the receiving node. If there was a collision, the array is cleared and the receiving node makes no response (CKRXCLN returns a value of FALSE).

```
FUNCTION CKRXCLN(NODE,MSGID,PKTID)
  CKRXCLN = FALSE
  FOR EACH SLOT
    IF ((RXCOLLISION(NODE,SLOT,MESSAGEINDEX) = MSGID) .AND.
      (RXCOLLISION(NODE,SLOT,PACKETINDEX) = PKTID)) THEN
      CKRXCLN = TRUE
    END IF
  END FOR
END CKRXCLN
```

MISCELLANEOUS

Subroutine GNEGNAK. Called by RECVPT when a data packet has an error and negative ACKs are specified in the model. It generates a negative ACK and enters it into the SLAM network for node processing.

```
SUBROUTINE GNEGNAK
  IF ((NEGATIVE ACKS ARE SELECTED).AND.(NOT RING NETWORK))
    THEN
      PACKET LENGTH = PACKET OVERHEAD
      DESTINATION = LAST NODE
      PACKET TYPE = NEGATIVE ACK
      ENTER INTO NETWORK NODE
    END IF
  END GNEGNAK
```

Subroutine GENACKS. Called by RECVPKT. It generates end-to-end ACKs for bus and mesh models and node-to-node ACKs for mesh models only.

```
SUBROUTINE GENACKS(CURRENT NODE, LAST NODE)
PACKET LENGTH = OVERHEAD
ERROR = NO
ASSIGN PACKET ID FROM ATTRIBUTES

IF (MESH NETWORK) THEN
    PACKET TYPE = NODE TO NODE ACK
    DESTINATION = LAST NODE
    ENTER INTO NETWORK FOR QUEUEING FOR TRANSMISSION
END IF

IF (AT DATA DESTINATION NODE) THEN
    PACKET TYPE = END TO END ACK
    DESTINATION = DATA ORIGIN NODE
    ENTER INTO NETWORK FOR QUEUEING FOR TRANSMISSION
END IF

END GENACKS
```

Subroutine ENTRRTX. Called by CLRSVRQ when there were no collisions at the transmitting node. ENTRRTX removes the packet from the transmitting node server queue. It then enters the packet into the SLAM network to begin the retransmission queueing and time-out process.

```
SUBROUTINE ENTRRTX(SERVERFILE, MSGID, PKTID)
RETRANSMITPACKET = POINTER TO PACKET IN SERVERFILE
IF (PACKET IS NOT ON FILE) THEN
    CALL ERROR(UNABLE TO FIND PACKET FOR RETRANSMISSION)
END IF
REMOVE PACKET FROM SERVERFILE
ERROR = FALSE
ENTER PACKET INTO THE SLAM NETWORK FOR THE RETRANSMISSION
PROCESS
END ENTRRTX
```

Subroutine CLRSVRQ. Called by subroutine FREERSC when a packet ends transmission. If no collisions were detected at the transmitting node, it clears the packet from the server queue at the transmitting node.

```
SUBROUTINE CLRSVRQ(NODE, MSGID, PKTID)
IF (NO COLLISION DETECTED AT TRANSMITTING NODE) THEN
    IF (DATA PACKET) THEN
        CALL ENTRRTX(SERVERFILE, MSGID, PKTID)
    ELSE
        CALL RMVEPKT(SERVERFILE, MSGID, PKTID)
    END IF
```

END IF
END CLRSVRQ

Subroutine STATCOL. Called by the SLAM network cyler for statistics collection. It collects several statistics at an interval that is determined by a SLAM XX variable.

```
SUBROUTINE STATCOL
NUMBEROFFPACKETS = TOTAL NUMBER OF OBSERVATIONS OF
    PACKET DELAY - LAST OBSERVATION OF NUMBEROFFPACKETS
COLLECT STATISTICS ON THE NUMBEROFFPACKETS
    SINCE LAST OBSERVATION
NUMBEROFFPACKETS = TOTAL NUMBER OF OBSERVATIONS OF
    PACKET DELAY
COLLECT STATISTICS ON COLLISIONS AND ERRORS
RESET COLLISION AND ERROR COUNTERS
END STATCOL
```

Subroutine PRTFIL. Called by the SLAM network to print out selected files. It is a diagnostic tool used only when required and tailored for each use.

```
SUBROUTINE PRTFIL
CALL PRNTE (ANY FILE)
CALL PRNTE (ANY FILE)
(REPEAT FOR ANY FILE OF INTEREST)
END PRTFIL
```

Subroutine PRTSTAT. Called by the SLAM network to print out selected statistics.

```
SUBROUTINE PRTSTAT
PRINT HEADER
NUMBER OF PACKETS = TOTAL NUMBER OF OBSERVATIONS OF PACKET
    DELAY - LAST OBSERVATION OF OF NUMBER OF PACKETS
PRINT THROUGHPUT (* NUMBER OF PACKETS *)
AVERAGE DELAY = TOTAL OF ALL DELAYS / NUMBER OF PACKETS
PRINT AVERAGE DELAY
DECREMENT MESSAGE INTERRIVAL RATE
END PRTSTAT
```

Subroutine CLRCAL. Called from subroutines RECVPKT and CSMAST. It is a special mechanism to remove events from the SLAM calendar. The retransmit time out activities and the node to node transmission stop activities cannot be stopped once started in the network except by direct calendar manipulation. Assignments are made in the SLAM network to selected attributes to tag these events with the message ID. RECVPKT removes retransmit time out activities and CSMAST removes transmission stop activities by passing the attribute number marked with the message ID.

```

SUBROUTINE CLRCAL(MSGID,PKTID,NODE,ATTRIBUTE)
POINTER = LAST ENTRY ON THE CALENDAR
WHILE((NOT EOF) OR.(NOT FOUND))
  CALENDAR MESSAGE = POINTER(ATTRIBUTE)
  CALENDAR PACKET = POINTER(14)
  CALENDAR NODE = POINTER(10)
  IF (((CALENDAR MESSAGE).EQ.MSGID).AND.
      (CALENDAR PACKET.EQ.PKTID)) .AND.
      (CALENDAR NODE.EQ.NODE)) THEN
    REMOVE EVENT FROM CALENDAR
  END IF
END WHILE

```

Appendix B

Quick Reference List of Specifications

<u>SPECIFICATION</u>	<u>IMPLEMENTATION</u>	<u>SELECTIONS</u>	<u>ENTRY POINT</u>
A. Global			
1. Topology type	XX(33)	distributed=1 ring=2 bus=3 fixed=1 adaptive=0 arbitrary time	INTLC statement
2. Routing (distributed topology only)	XX(53)		INTLC statement
3. Ring options	XX(54)		INTLC statement
a. node latency	XX(18)	integer	INTLC statement
b. network access scheme	XX(38)	token=1 slotted=2	INTLC statement
4. Bus options			
a. CSMA/CD	XX(51)	on=1, off=0	INTLC statement
b. network access scheme	XX(47)	random delay=1 slotted=2	INTLC statement
1. slot length	XX(50)	integer	INTLC statement
2. max propagation delay on bus	XX(52)	real	INTLC statement
3. minimum packet spacing	XX(48)	real	INTLC statement
5. Message Protocols			
a. message length	USERF(7)	constant or distribution	USERF
b. maximum message length	XX(25)	integer (distrib. average)	INTLC statement
c. maximum data packet length	XX(21)	integer (must use USERF(7))	INTLC statement
d. packet overhead	XX(23)	integer	INTLC statement
e. compression function	XX(24)	integer	INTLC statement
	XX(29)	0.0-1.00	INTLC statement

f. message arrival process	XX(20)	(1.00=no compression) interarrival rate=1 last message=0	INTLC statement
6. Link protocols			
a. negative acks	XX(31)	yes=1 no=0	INTLC statement
b. Retransmit time-out			
1. most models	XX(14)	real	INTLC statement
2. non-CSMA/CD bus models	XX(56) XX(55)	real (hi and low ends of UNIFRM dist.) integer	INTLC statement INTLC statement
c. Hold time-out	XX(13)	yes=1 no=0	INTLC statement
d. Immediate error detection and response	XX(32)		
7. Flow control			
a. minimum buffer level for message creation	XX(36)	integer (buffer)	INTLC statement
b. max # of packets queued for node	XX(45)	integer	INTLC statement
c. minimum buffer level required after packet admission	XX(37)	integer	INTLC statement
d. max # of packets in node for a given node	XX(46)	integer	INTLC statement
e. priority scheme		yes=priority statement no=no statement integer	PRIORITY statement INTLC statement
f. node server queueing maximum	XX(44)		
8. Propagation delay of media (time/dist.)	XX(16)	real or integer (may be 0)	INTLC statement
9. Error Function (probability of error)	XX(30)	0.0-1.00 0.0=>no error	INTLC statement
10. Encryption rate (data unit/time unit)	XX(10)	real	INTLC statement

B. Host Options				
1. message interarrival time (use in network)	USERF(17) and XX(17) or USERF(18) and XX(17)	real constant	USERF(I)	
2. Message regeneration control			USERF(I)	
a. delay after packet acceptance	XX(15)	arbitrary delay	USERF(I)	
b. delay after creation denial	XX(39)	arbitrary delay	INTLC statement	
3. host-node transfer rate (data/time unit)	XX(11)	real or integer	INTLC statement	
4. node to host delay (time)	XX(57)	real	INTLC statement	
5. destination distribution		random selection (based on N nodes)	MAKPKTS	
C. Node Options				
1. Number of nodes	XX(26)	1-9	INTLC statement	
2. Maximum number of hosts		N-29	SLAM Network	
3. Number of lines per node	XX(27)	0-5 (total 20)	INTLC statement	
4. For each line				
a. transmission rate	XX(12)	data units/time unit	INTLC statement	
b. physical length	input	real or integer	Input File	
c. destination node (node numbers)	input	1 to N	Input File	
5. buffer capacity of each node	XX(35)	integer (homogeneous assignment)	INTLC statement	
	XX(1-9)	integer (non-homogeneous assignment)	INTLC statement	
D. Administrative				
1. message ID counter	XX(22)	100	INTLC statement	
2. instrumentation switches	XX(40-43)	0 or 1	INTLC statement	

3. calendar delay time	XX(49)	0.00000001	INTLC statement
4. transmit collision counter	XX(60)	0	INTLC statement
5. receive collision counter	XX(61)	0	INTLC statement
6. transmission error counter	XX(62)	0	INTLC statement
7. statistics collection interval	XX(65)	arbitrary time	INTLC statement
8. packet counter	XX(66)	0	INTLC statement
9. delay time accumulator	XX(67)	0	INTLC statement
10. interarrival rate change	XX(68)	arbitrary rate change	INTLC statement

Appendix C

Demonstrations of Computer Network Models

This appendix shows how several models were developed for demonstration purposes. Some analysis and discussion are also included for each model.

The first section describes the administrative specifications that must be made for most models. (Specific discussion of some of them are omitted in the mesh, ring, and bus models which follow.)

I. Administrative Specifications

- A. $XX(22) = 100$. The message ID counter for the program is initialized to 100. Lower numbers are not used to allow calendar manipulation without interfering with valid message ID's.
- B. $XX(40-43) = 0$. Instrumentation switches are normally left off.
- C. $XX(49) = 0.00000001$. A very small value is used to delay selected activities to insure the correct ordering of EVENTS on the SLAM calendar.
- D. $XX(60, 61, 62) = 0$. Counters for collisions at the transmit end and receive end, and transmission errors are initialized.
- F. $XX(65) = \text{variable}$. This statistics collection interval may differ from model to model and from one run to another. It should be chosen large enough to insure the system is in steady-state for most of the interval. It should be zero if only the SLAM SUMMARY statistics are desired.
- G. $XX(66) = 0$. This initializes the packet count.
- H. $XX(67) = 0$. This initializes the accumulator of delay times.
- I. $XX(68) = \text{variable}$. This change to the message inter-arrival rate also may be changed dynamically. It is used in conjunction with the statistics collection interval to generate different message traffic intensities during a simulation run. The system is not re-initialized in any way for each of these intervals.
Note: The SLAM SUMMARY Report is based on entire runs. When the system is changed in this way during a run, the SLAM statistics may be of little or no value except to show some of the current conditions at the end of the run.

II. Demonstration of mesh model. A five-node model was developed using the topology shown in Figure 14. Parameters were chosen to approximate those found in computer-communications networks such as those found in the ARPA network. The following development of the model follows the outline of Chapter IV and Appendix B.

MODEL DEVELOPMENT

A. Global Specifications

1. XX(33) = 1. Mesh model
2. XX(53) = 1. Fixed routing
- 3 and 4. N/A
5. Message Protocols
 - a. USERF(7) and XX(25) = 800. Message length from an exponential distribution using a mean of 800 bits.
 - b. XX(21) = 8000. Maximum length of message (bits)
 - c. XX(23) = 1000. Maximum length of packet (bits)
 - d. XX(24) = 24. Packet overhead
 - e. XX(29) = 0. No data compression
 - f. XX(20) = 1. Message arrival process determined by an interarrival rate.
6. Link Protocols
 - a. XX(31) = 0. No negative acknowledgements
 - b. XX(14) = 125. 125 msec retransmit time-out
 - c. XX(13) = 1000. 1000 msec hold time-out
 - d. XX(32) = 0. No immediate detection of errors
7. Flow Control
 - a. XX(36) = 30000. If less than 30000 bits of buffer are available at a node, it's hosts may not create new messages.
 - b. XX(45) = 4. If 4 or more packets are already queued for entry to a node, its hosts may not create a new message.
 - c. XX(37) = 20000. If less than 20000 bits of buffer are available at a node, arriving packets from the hosts will be queued for entry to that node.
 - d. XX(46) = 12. If 12 or more packets are already admitted into a node for a given destination, arriving packets will once again be queued for entry.
 - e. Priority statements. Acknowledgements have priority in server files and files that represent queueing for node entry.
 - f. XX(44) = 1000. A high number to allow queueing for each server at each node.
8. XX(16) = 0.005. Propagation delay

9. XX(30) = 0.0 to 0.05. Packet error rates over the runs vary.
 10. XX(10) = 10000. Encryption rate (bits/msec)
- B. Host Options
1. USERF(18) and XX(17) = 10. USERF(18) is used in network host structures to produce an interarrival times based on a mean of XX(17). (XX(17) was varied over the runs.)
 2. N/A
 3. XX(11) = 10000. host-node transfer rate in bits/msec.
 4. XX(57) = 1.0. An arbitrary (msec) delay between the time when a node has a message ready and when a host will accept it.
 5. Random selection of message destination. This is currently performed in subroutine MAKPKTS and is based on the number of nodes.
- C. Node Options.
1. XX(26) = 5. Five nodes in the network
 2. A total of ten hosts were created (two per node).
 3. XX(27) = 3. A maximum of three lines per host was used in this model.
 4. Line parameters
 - a. XX(12) = 960. This bit/msec transmission rate was chosen a factor of 10 lower than a typical 9600 baud rate to build a model that is less demanding in terms of computer resources. For the faster rate a very large number of packets would be required in the server queues to test the system. Previous trial runs indicate that delay times should be a factor of 10 higher with this rate.
 - b. and c. Reference Figure 14 for node-to-node distances and destination node for each line.
 5. XX(35) = 40000. Each node has 40000 bits of buffer available. INTLC assigns this value to XX(n) for each node n.

ANALYSIS

There are many things that could be analyzed with this model. In the interest of demonstrating the model, a comparison was made of the system behavior in conditions of different transmission error rates. The simulation was first run with no transmission errors. Runs were then made with a packet error rate of 5% (five percent of the

packets had errors simulated in them.) The simulation runs were divided into intervals in which progressively shorter interarrival times were used for all ten host message generation processes. The interarrival times started at ten msec and were decreased by two msec each interval to two msec for the last interval.

The length of each statistics collection interval with a constant interarrival time must be chosen to insure an equilibrium condition has been reached. To test that assumption, different interval lengths were tested. The runs shown in Figures 32 and 33 were with 300 or 500 msec of simulated time for each interval.

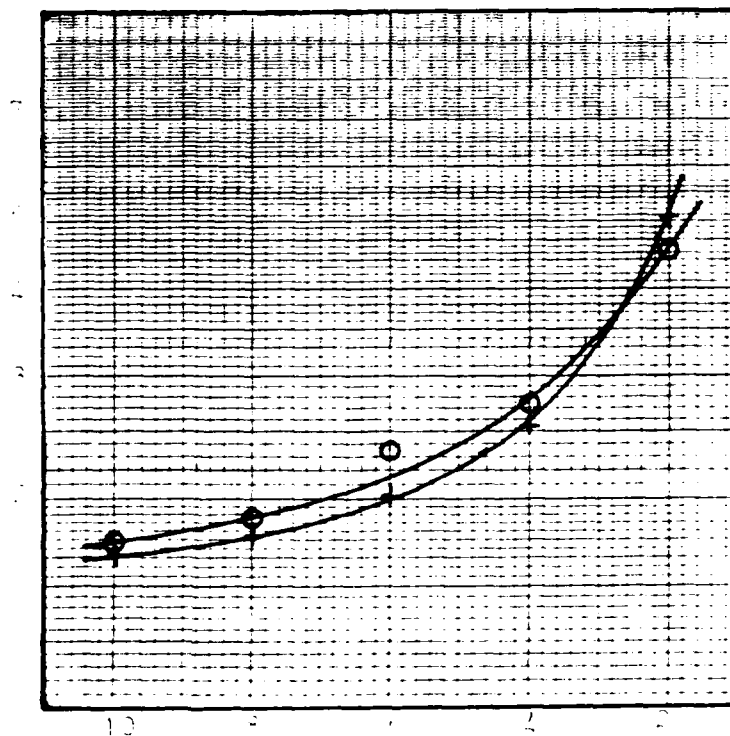
To take a cursory look at throughput, a plot against offered traffic is given in Figure 34. The behavior of the system may not be intuitive, therefore some comments are given concerning throughput and delay behavior in the discussions to follow.

DISCUSSION

The two curves of Figure 32 show behavior which is typical of computer networks. They overlap fairly well, which is a good indication that a steady-state was reached for the conditions imposed on the model (transmission error free).

The data for the runs plotted in Figure 33 do not support the assumption that a steady-state was reached. They don't show well-defined trends and do not overlap.

Delay
Interval

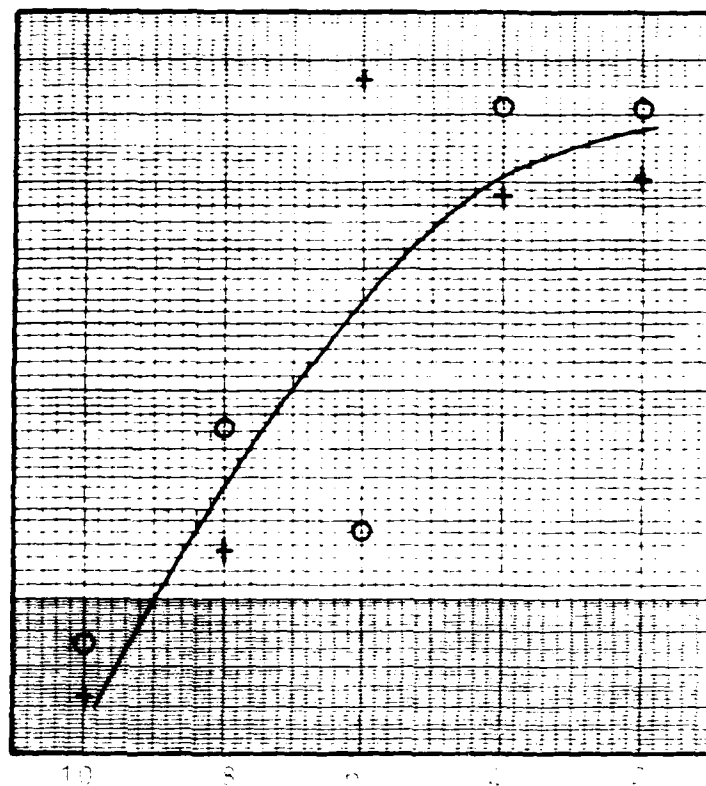


Average Interarrival Time
at each Host (used)

- O - 300 msec/Interval
- + - 500 msec/Interval

Figure 32. Moch Model Delay with 2% Error Rate.

Delay
msec



Average Interarrival Time
at each Host (msec)

○ - 300 msec/Interval
+ - 500 msec/Interval

Figure 33. Mesh Model Delay with 5% Error Rate.

AD-A127 318

DESIGN AND IMPLEMENTATION OF A GENERIC COMPUTER NETWORK
SIMULATION SYSTEM(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING

33

UNCLASSIFIED

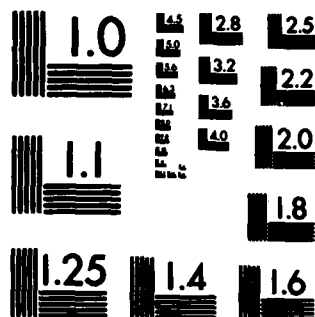
S J FOSTER MAR 83 AFIT/GCS/EE/83M-2

F/G 9/2

NL

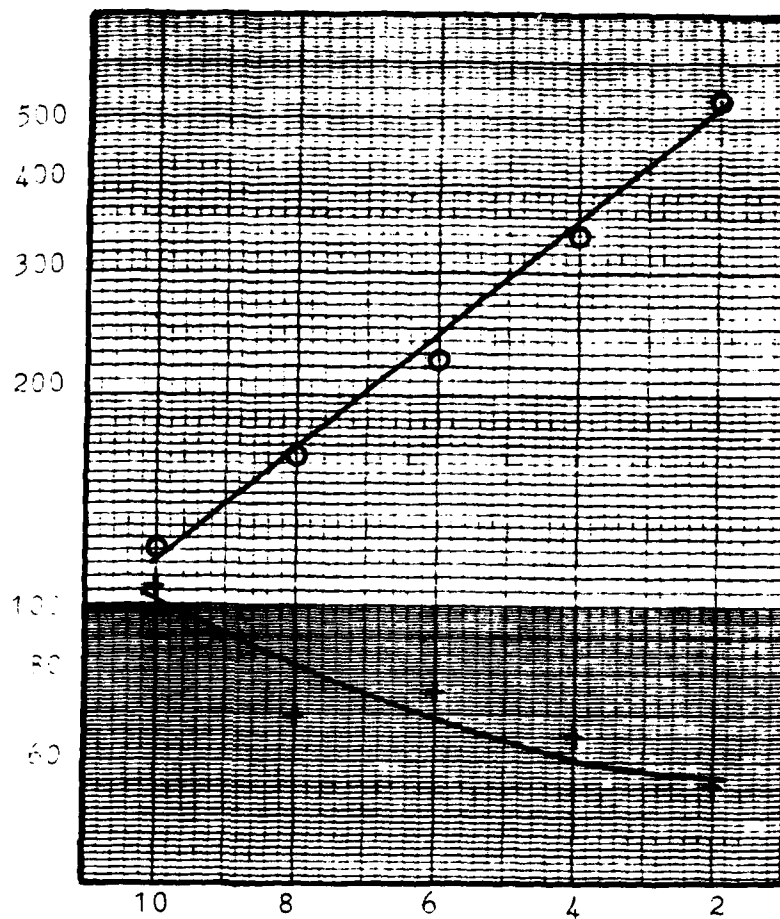


END
DATE
FILMED
5 83
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Throughput
(Packets/
100 msec)



Average Interarrival Times
at each Host (msec)

○ - 0% Error Rate

+ - 5% Error Rate

Figure 34. Mesh Model Throughput.

(The curve drawn in is a rough approximation of the delay behavior.)

When errors occur, a very long retransmission time-out must expire before packets may be retransmitted. Since this time-out is two orders of magnitude greater than a data transmission period, a small number of errors can cause a large difference in average delay. Therefore, to establish the limiting behavior of this system, much greater statistics collection intervals must be used. (The run with 500 msec/interval generated only 1870 observations of delay in a total of 2500 msec.)

The direction of inflection of the curve drawn in at the top may be a matter of interest also. A queueing system perspective may lead one to expect delay times to approach infinity. The system, as simulated, has limits on the number of packets it can hold and queue which are enforced by flow control criteria. So, as the system reaches a certain point, it accepts no more packets, but serves only what is in the system until the system flushes itself out. The interaction of the time-outs in the retransmit and hold queues, and server queueing times will then determine the system behavior.

In conditions free of transmission errors, the throughput increases as expected with decreasing inter-arrival times (Figure 34). However, with a 5% packet error rate the throughput decreases with increasing offered traffic. As more traffic is offered the system

becomes clogged with packets in the hold and retransmit queues. At some point, flow control criteria are reached and external sources are shut off. The system will then allow only the traffic already in the system to proceed. That traffic will proceed very slowly as the time-outs expire. As a result, little traffic is allowed to trickle in as the system slowly clears. The retransmission time-outs and flow control criteria will then determine the resulting throughput, not the arrival rate of external traffic.

The SLAM SUMMARY reports for the mesh model in runs of constant interarrival time contain a wealth of information. (An example of the output from a mesh network is included in the program and documentation package available from Maj Seward.) The dynamics of the system can be analyzed in great detail. The discussion offered above is a very small part of a more thorough analysis.

The data supporting the preceding discussion was taken from four simulation runs of different lengths and statistics intervals. On the average, an interval of 500 msec required 29 seconds of execution.

- III. Demonstration of token ring model. This model was chosen from Bux (4) to provide data for comparative analysis with some published data. The objective was to add validity to the program even though the token ring network is

not found in computer-communication networks. It is found in smaller local-area networks.

MODEL DEVELOPMENT

A. Global Specifications

1. XX(33) = 2. Ring model
2. N/A
3. Ring Options
 - a. XX(18) = 8. Eight bits of latency for each of the five nodes was chosen to approximate the 50 nodes with one bit latency.
 - b. XX(38) = 1. Selection of token ring access scheme.
4. N/A
5. Message Protocols
 - a. USERF(7) and XX(25) = 1000. USERF(7) gives message lengths from an exponential distribution with a mean of 1000 bits.
 - b. XX(21) = 5000. An upper limit on message length.
 - c. XX(23) = 5000. The maximum data packet length was chosen the same as the message length to insure all messages were single packets as inferred in Bux's paper.
 - d. XX(24) = 24. Packet overhead to match 24 bit header.
 - e. XX(29) = 1.0. No data compression simulated.
 - f. XX(20) = 1. Message interarrival process based on a rate.
6. Link Protocols
 - a. XX(31) = 0. No negative acknowledgements.
 - b. XX(14) = 100. Retransmit time-out in msec.
 - c. N/A
 - d. N/A
7. Flow Control
 - a. XX(36) = 10000. If less than 10000 bits of buffer are available at a node, it's hosts may not create new messages.
 - b. XX(45) = 5. If five or more packets are already queued for entry to a node, it's hosts may not create new messages.
 - c. XX(37) = 10000. If less than 10000 bits of buffer are available at a node, arriving packets will be queued for entry to that node.
 - d. XX(46) = 5. If five or more packets are already admitted into a node for a given destination, arriving packets for that destination will be queued for entry to the node.
 - e. No priority statements
 - f. XX(44) = 5. The number of packets queued for a server was restricted because Bux's 50 nodes did not queue packets.

8. $XX(16) = 0.005$. Propagation delay used by Bux (msec/km).
 9. $XX(30) = 0$. No errors were simulated.
 10. $XX(10) = 10000$. Encryption rate (bits/msec)
- B. Host Options
1. $USERF(18)$ and $XX(17) = 25$. $USERF(18)$ was used in the SLAM network to give message interarrival times from an exponential distribution with mean given by $XX(17)$. ($XX(17)$ was varied during the simulation run.)
 2. N/A
 3. $XX(11) = 10000$. Host-node transfer rate in bits/msec
 4. $XX(57) = 0$. No delay for node to host transfer
 5. Random selection of message destination
- C. Node Options
1. $XX(26) = 5$. Five nodes in the network.
 2. A total of ten hosts were created (two per node).
 3. $XX(27) = 1$. One line per host
 4. Line parameters
 - a. $XX(12) = 1000$. This bit/msec transmission rate matches Bux's 1,000,000 bits/msec rate.
 - b. The physical lengths of all transmission lines were 0.4 km.
 - c. Destination node for each line was the next node in the ring.
 5. $XX(35) = 60000$. Each node has 60000 bits of buffer available.

ANALYSIS

Figure 35 is a plot of delay versus throughput. Both axes are normalized as indicated as in the Bux paper. The solid line is directly from Bux and the data points are from the simulation using the model developed above. No attempt to draw a curve for the demonstration was made. (Note the comments in the discussion to follow.)

DISCUSSION

The simulation data points generally follow the curve which is drawn in from the Bux paper. The simulation data points are somewhat scattered around the curve. Longer statistics collection intervals could be used to even out the data and insure a good representation of steady-

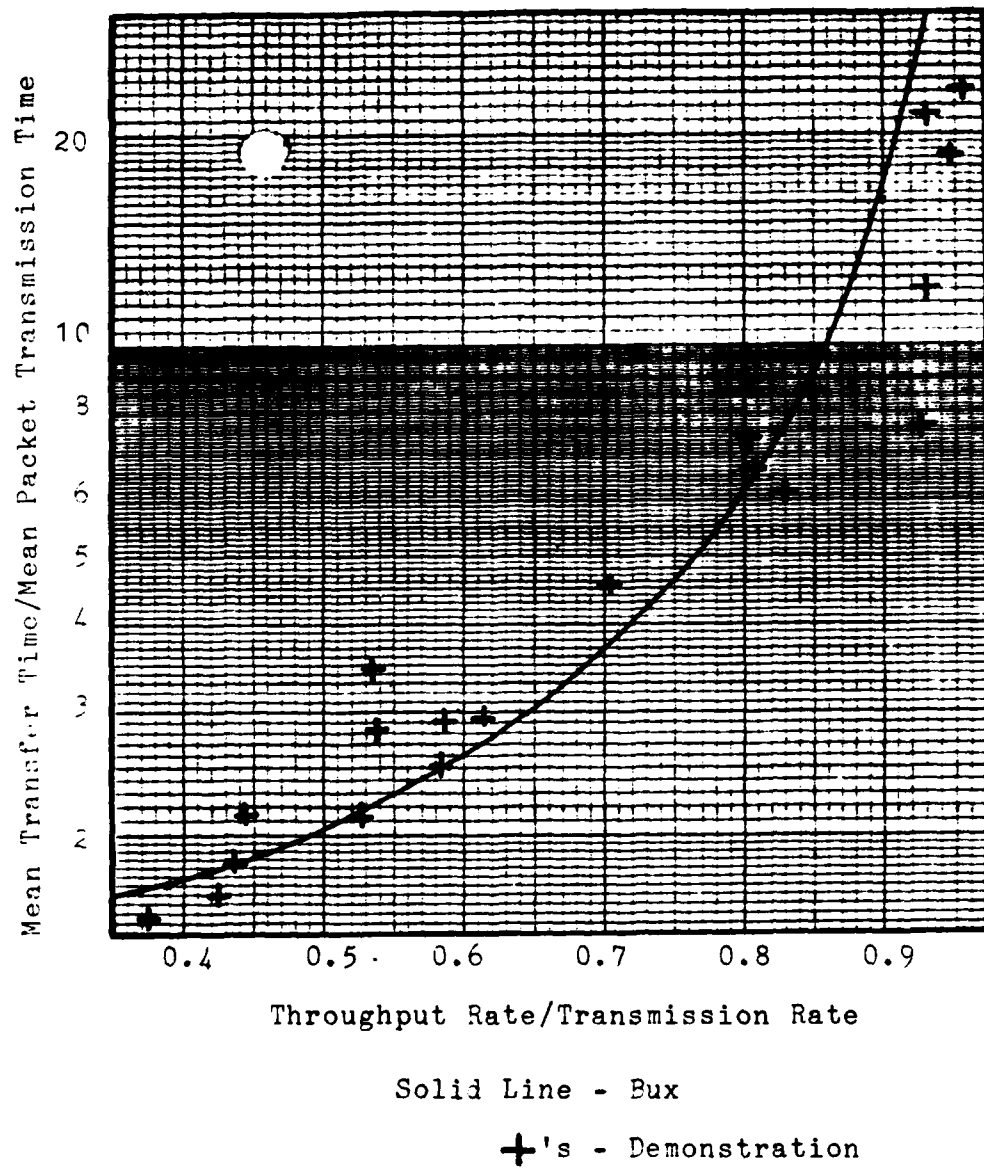


Figure 35. Delay vs Throughput For Token Ring Model.

state. However, the token ring models the packet transmission process through every node for all packets. In addition, an empty token travelling around the ring generates many EVENTS in a short period of time. As a result the amount of computer time required to simulate the model is very great. For this run the Cyber computer consumed almost 800 seconds of execution time to simulate 9.5 seconds of simulation time.

IV. Demonstration of the slotted ring model. This model also appeared in the Bux paper. Several specifications required careful consideration to demonstrate this network.

MODEL DEVELOPMENT

A. Global Specifications

1. XX(33) = 2. Ring model
2. N/A
3. Ring Options
 - a. XX(18) = 10. Ten bits of latency per node was critical in this model. The cumulative node latency plus the propagation delay around the bus was designed to equal the slot length.
 - b. XX(38) = 2. Selection of the slotted ring model
4. N/A
5. Message Protocols
 - a. USERF(7) and XX(25) = 50000. USERF(7) returns a message length from an exponential distribution with mean XX(25). It also truncates the value to a maximum given by XX(21). An average message length of 50,000 was given to force most message lengths to the slot length.
 - b. XX(21) = 36. Maximum message length (also slot length)
 - c. XX(23) = 36. Packet length to insure all single-packet messages.
 - d. XX(24) = 24. Packet overhead to match 24 bit header.
 - e. XX(29) = 1.0. No data compression
 - f. XX(20) = 1. Message interarrival process based on a rate.

- 6. Link Protocols
 - a. $XX(31) = 0$. No negative acknowledgements
 - b. $XX(14) = 100$. Retransmit time-out in msec
 - c and d. N/A
- 7. Flow Control
 - a. through f. Identical with token model.
- 8. $XX(16) = 0.005$. Propagation delay used by Bux (msec/km). (The input file of node-to-node distances, this parameter, and the node latency had to be designed to produce the correct slot length.)
- 9. $XX(30) = 0$. No errors were simulated.
- 10. $XX(10) = 10000$. Encryption rate (bits/msec)
- B. Host Options
 - 1. through 5. Identical with token model.
- C. Node Options
 - 1. through 4. Identical with token model.
 - 5. $XX(35) = 40000$. Each node has 40000 bits of buffer available.

ANALYSIS

Figure 36 is a plot generated in the same manner as the token model (Figure 34). The solid line is directly from Bux and the data points are from the simulation run using the model developed above. A curve was approximated from the data points. The curve was not extended fully to the right because few data points were available in that area.

DISCUSSION

The demonstration follows the Bux curve fairly well after two adjustments were made to the normalized delay times. First, one slot length had to be added to the delay time. For this model, the packet delay is based on the arrival of the leading edge of the slot at the destination node. The slot (or packet) doesn't really arrive until the trailing edge arrives one slot length later.

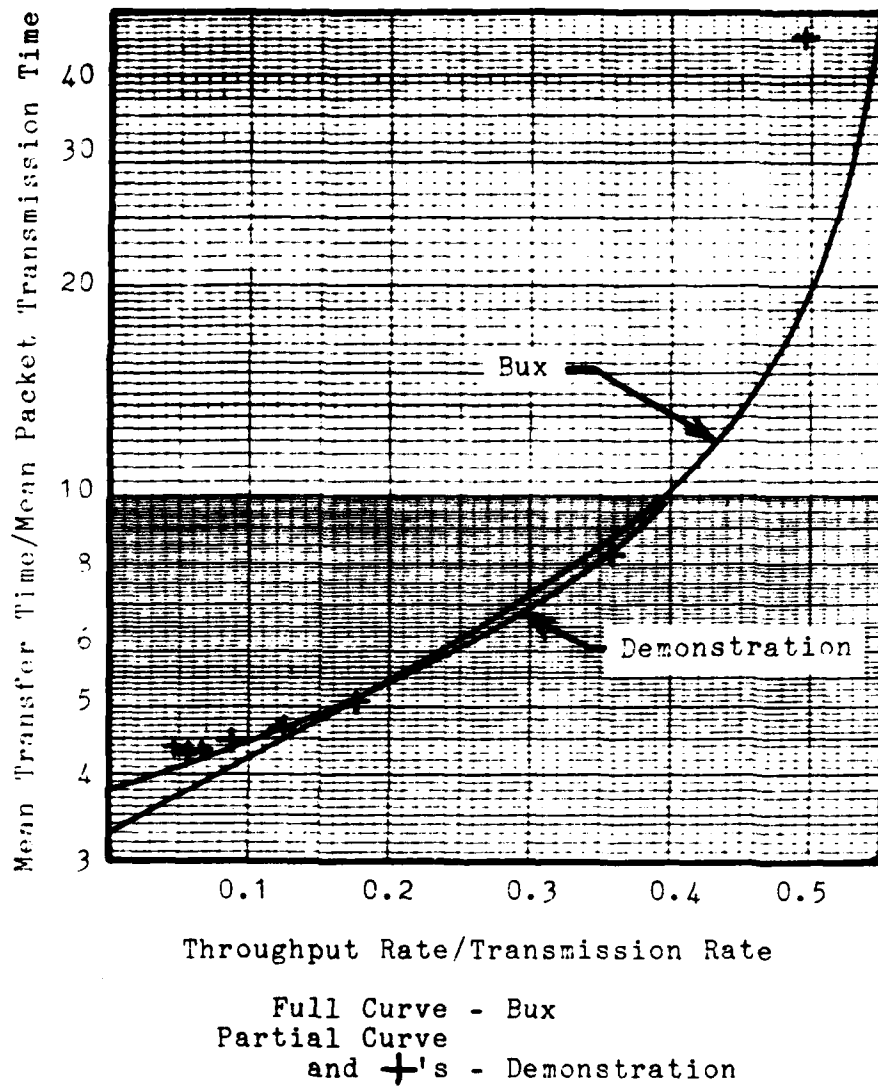


Figure 36. Delay vs Throughput for Slotted Ring Model.

The second adjustment is one-half of a slot length for a complete transmission of a slot around the ring. The transmission (in this slotted case) is not complete until the slot is received at the node that transmitted the data. In this program, delay is based on packet arrival at the destination. On the average, the destination node is one-half of the ring away from the origin node. Therefore, to match the Bux model, another one-half of a slot must be added to the delay.

The data points at the higher throughput levels are not very dense. The amount of data moved per slot is very small in this model. Therefore large numbers of packets are required to generate throughput at the upper limits of the model. In turn, many EVENTS are required which is reflected in execution times one hundred times greater than simulated times. From the data that was generated, the model was validated within a sufficient level of confidence.

- V. Demonstration of a CSMA/CD Bus network. The bus modeled here is similiar to the CSMA/CD bus in Bux. This model is significantly different in several respects. There are five nodes instead of fifty. Additionally, the fifty nodes of Bux were not allowed to queue messages. To approximate that characteristic, queueing allowed in the five nodes of this simulation model was restricted. Also,

the Bux model did not use acknowledgements which are part of this simulation model.

MODEL DEVELOPMENT

A. Global Specifications

1. XX(33) = 3. Bus model
2. N/A
3. N/A
4. Bus options
 - a. XX(51) = 1. CSMA/CD selected
 - b. XX(47) = 1. Network access scheme based on a random delay.
 1. N/A
 2. XX(52) = 0.02. Maximum propagation delay on the bus (msec).
 - c. XX(48) = 0.05. Minimum packet spacing (msec)
5. Message Protocols
 - a. USERF(7) and XX(25) = 1000. USERF(7) was used to return message lengths from an exponential distribution with a mean of 1000 bits.
 - b. XX(21) = 5000. Maximum message length to include 95% of messages from above distribution without truncation.
 - c. XX(23) = 5000. Packet length chosen to provide single-packet messages
 - d. XX(24) = 24. Packet overhead to match 24 bit header
 - e. XX(29) = 1.0. No data compression
 - f. XX(20) = 1. Message interarrival process based on a rate.
6. Link Protocols
 - a. XX(31) = 0. No negative acknowledgements
 - b. XX(14) = 120. Retransmit time-out in msec
 - c. and d. N/A
7. Flow Control
 - a. through f. Identical with token and slotted ring models.
8. XX(16) = 0.005. Propagation delay used by Bux (msec/km)
9. XX(30) = 0.0. No errors were simulated.
10. XX(10) = 10000. Encryption rate (bits/msec)

B. Host Options

1. USERF(18) and XX(17) = 50. USERF(18) was used in the SLAM network to give message interarrival times from an exponential distribution with a mean given by XX(17). (XX(17) was varied during the simulation run.)
2. N/A
3. XX(11) = 10000. Host-node transfer rate in bits/msec.
4. XX(57) = 0. No special delays for node to host transfer.
5. Random selection of destination distribution

C. Node Options

1. XX(26) = 5. Five nodes in the network
2. A total of ten hosts were created (two per node).
3. XX(27) = 1. One line (output) per node
4. Line parameters
 - a. XX(12) = 1000. Transmission rate (bits/msec)
 - b. The physical lengths between nodes were randomly chosen with none exceeding the 2 km "cable length" of Bux.
 - c. N/A
5. XX(35) = 50000. Each node has 50000 bits of buffer available.

ANALYSIS

Figure 37 is a plot of delay versus throughput. The Bux curve is drawn in and the demonstration curve is estimated from the data points. Axes are normalized as in the previous ring models.

DISCUSSION

In the Bux paper normalized delay for the CSMA/CD bus rises rapidly to 100 at a normalized throughput just above 0.9. The model used in this demonstration displays the same general behavior, but reaches a maximum normalized throughput of approximately 0.79. Additionally, the normalized delay is consistently higher in the demonstration. The most obvious reason for this is the acknowledgement scheme used in this model.

The difference in maximum throughput between the two models can be easily understood. In a simplified argument, the time domain of the bus can be divided into four components: data transmission, overhead transmission, bus arbitration (including collisions and thus wasted time), and idle time. Since the following argument is concerned with the limiting behavior of the bus, idle time (where

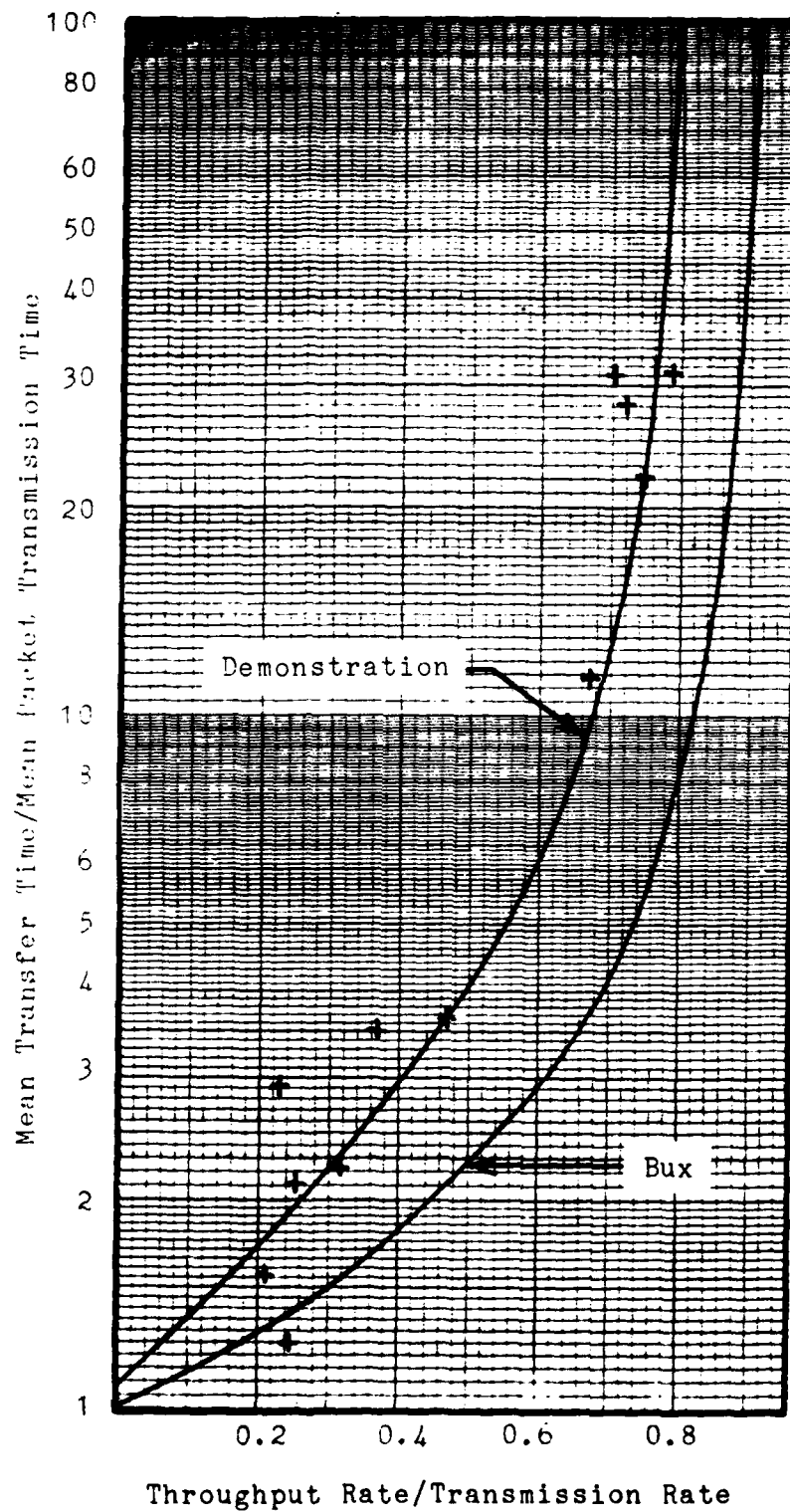


Figure 37. Delay vs Throughput for CSMA/CD Bus

no nodes have data and are thus not attempting transmission) is not considered. In its limiting behavior, the Bux model spent 90% of the time available transmitting data. That leaves 10% for overhead and bus arbitration time. Given the 1000 bits average message size, 1000 bits/msec transmission rate, a 24 bit header, and 90% of the time in data transmission, approximately 2.1% (or 2%) of the bus time was spent in transmitting overhead. That leaves 8% for arbitration.

The demonstration model using acknowledgements requires two more elements, transmission of the acknowledgement, and the arbitration of the bus for that transmission. Since the acknowledgement is also 24 bits, it should also take about 2% of the bus time. An additional 8% of the bus time will be required for arbitration for the acknowledgements. That leaves (with this very simplified argument) approximately 80% of the time left for data transmission. Since the bus demonstration gave 79% for its maximum normalized throughput, the prediction was relatively accurate. Small subtleties in the lower level details of the models could have contributed to the small difference. Additionally, variations in the random processes of the simulation could also have contributed to the difference.

The greater delays for the demonstration model can also be attributed to the added time for the acknowledgement. A queueing model to predict this behavior would re-

quire adding complexities such as idle times and queueing times. Since modeling and simulation are the primary emphasis of this thesis, a queueing model for this case would be beyond the scope of this thesis.

Other behavior not apparent from the plot should also be pointed out. After reaching the maximum throughput, the demonstration model's throughput dropped off. I will offer an explanation, which can be considered a precaution. Severe loading can produce conditions in which the "normal" arbitration schemes become ineffective. Therefore, the arbitration scheme should change dynamically in response to such conditions. An elegant dynamic scheme was not implemented for this demonstration. A simple implementation of the binary exponential back-off algorithm (18) was used for the delay between transmission attempts. A more elaborate scheme may be necessary to maintain high throughputs.

The average execution time for a 500 msec interval of simulation was 6 seconds. This was the most inexpensive of the four models to run.

APPENDIX D

General Legend of SLAM Variables

ATTRIBUTE LEGEND

- 1 message generation time
- 2 origin node
- 3 message length
- 4 message id number
- 5 destination node
- 6 destination host (used by ringbak and freersc
for data destination)
- 7 packet length
- 8 end of message packet 1=last 0=other
- 9 packet arrival at node (origin time)
- 10 present node location
- 11 packet type 1=data 2=ack 3=e-eack 4=nack
5=ring ack
- 12 next node
- 13 routing/branching indicator
- 14 packet sequence #
- 15 error indicator 1=error 0=ok
- 16 origin host (also used for tagging calendar
events that represent the network
call to subroutine STOPACT
- 17 branch flag 1=transmit 0=go on queue
(also used for tagging entities
for the subroutines RETXMT and
HLDOUT
- 18 data destination

SLAM global XX() VARIABLES

- 1-N buffer capacity of network nodes (N=number of nodes)
- 10 encryption rate
- 11 host to node transfer rate
- 12 node to node transmit rate
- 13 hold time out
- 14 retransmit time-out
- 15 time delay following admission of the last packet
into a node. (when XX(20)=0 only)
- 16 propagation delay (time units/node to node distance
units)
- 17 average interarrival rate
- 18 ring node latency
- 19 RFU
- 20 unrestricted message generation 1=unrestricted
0= after last message accepted
- 21 max message length
- 22 message id counter and generator
- 23 max data packet size

24 packet overhead
 25 average message length
 26 number of nodes
 27 number of lines (ports)
 28 RFU
 29 compression function
 30 probability of error in transmission
 31 negative acks: 1=yes 0=no
 32 immediate detection of errors 1=yes 0=no
 33 major topology type: 1=mesh, 2=ring, 3=bus
 34 RFU
 35 buffer capacity of network nodes (initial and max)
 (homogeneous assignment)
 36 minimum resource level following message creation
 37 minimum resource level following packet admission
 from host
 38 ring controller switch 1=token 2=slotted
 39 time delay for message creation after the last
 message creation was suppressed
 40-43 instrumentation switches
 44 maximum # of packets in a server queue for packet
 admission into a node. (ring and bus models only)
 45 max number of packets queued for entry into a node
 (for message creation denial)
 46 max number of packets in a node for a common
 destination (to allow packet admission from host)
 47 bus arbitrator selector 1=random delay scheme
 2=slotted scheme
 48 minimum packet spacing
 49 time delay to control the SLAM event calendar
 50 slot length for bus
 51 CSMA/CD selector 1=selected 0=off
 52 maximum propagation delay for bus models
 53 queue length waiting factor for adaptive routing
 54 routing table update interval
 55 low value of uniform distribution used in USERF 19
 for bus network backoff delay
 56 high value for same distribution.
 57 node host delay
 58 RFU
 59 RFU
 60 counter for collisions at transmitting node
 61 counter for collisions at receiving node
 62 counter for transmission errors
 63 RFU
 64 RFU
 65 time interval for statistics collection
 66 packet count holder
 67 delay time accumulator
 68 interarrival rate change (per statistics interval)
 69 RFU
 70 token tx time

ERROR CODES

3000 node argument out of range for seizing resource
3001 request to seize negative resources
3002 attempt to seize beyond resource capacity
3003 popping empty stack
3004 stack overflow
3005 error in routing. exceeded number of lines
3006 node argument out of range for freeing resources
3007 attempt to free negative resources
3008 attempt to free resources beyond capacity
3009 packet not in server queue
3010 packet not in retransmitt queue
3011 used immediate error detection in non-mesh model

FILES

<u>INDEX</u>	<u>NUMBER</u>	<u>FUNCTION</u>
01 to N	one per node	hold file
11 to 1N	one per node	await encryption at host
21 to 2N	one per node	await transfer to node
31 to 3N	one per node	retransmitt file
41 to 40 + S	one per line	server queue
61 to 6N	one per node	reassembly file
71 to 70 + H	one per host	await transfer to host

N => number of nodes
S => number of servers
H => number of hosts

VITA

Stewart J. Foster was born on 30 July 1951 in Syracuse, New York. He graduated from East Syracuse-Minoa Central High School in 1969 and attended S.U.N.Y. College of Environmental Science and Forestry at Syracuse University. He received the degree Bachelor of Science in Wood Products Engineering in 1973. He was commissioned at the same time through Air Force Reserve Officers Training Corps and entered active duty in June 1973. His career began with assignments to three USAF radar stations where he served as Communications-Electronics Maintenance Supervisor and as Commander. He served as Maintenance Control Officer with the 21st NORAD Region at Hancock Field, N.Y. until entering the School of Engineering, Air Force Institute of Technology, June 1981. He is a member of the IEEE Computer Society and Tau Beta Pi.

Permanent address: Box 154-A

Constantia, New York 13044

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/83-2	2. GOVT ACCESSION NO. AD-4127 18	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN AND IMPLEMENTATION OF A GENERIC COMPUTER NETWORK SIMULATION SYSTEM		5. TYPE OF REPORT & PERIOD COVERED M.S. Thesis
7. AUTHOR(s) Stewart J. Foster Lieut USAF		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/DNA) Wright-Patterson AFB, Ohio 45433		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE March 1983
		13. NUMBER OF PAGES 201
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES Approved for public release; LAW AFB 83-12. John E. Wolaver Dept for Research and Professional Development Air Force Institute of Technology (ATC) Wright-Patterson AFB OH 45433 7 APR 1983		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Simulation, modeling, packet switching, simulation language, computer network		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) → A generic approach was used in modeling and simulating computer networks. The primary type of computer networks of interest in this study are characterized by a communications sub-network of nodes which serve host processors. Local area networks are also considered and may be modeled with this program. All models included packet switching and can be characterized as having distributed, ring or bus topology. The top level of the simulation program design is as general as		

DD FORM 1473

1 JAN 75

EDITION OF 1 NOV 68 IS OBSOLETE


UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

possible. The lower levels of the design are the building blocks of particular models. The simulation program was implemented with Simulation Language for Alternative Modeling (SLAM). The network and discrete event orientation of SLAM were combined in this simulation system. In general, the SLAM network portion models the computer network components and the Fortran subroutines provides details which define the protocols of the model. Four computer networks were modeled to demonstrate the simulation system. The system is very general. However, many networks may not be modeled precisely enough for formal validation without further development. Further development of simulation systems such as this should be in the discrete event orientation.



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

